VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

APLIKACE PRO RUČNÍ KONTROLU AUTOMATICKY ZPRACOVÁVANÝCH DAT

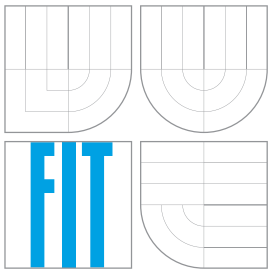
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ZDENĚK COUFAL

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

APLIKACE PRO RUČNÍ KONTROLU AUTOMATICKY ZPRACOVÁVANÝCH DAT

APPLICATION FOR HAND CORRECTIONS OF AUTOMATICALLY PROCESSED DATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ZDENĚK COUFAL
AUTHOR

VEDOUCÍ PRÁCE — Ing. IGOR SZÖKE, Ph.D.
SUPERVISOR

BRNO 2012

Abstrakt

Cílem této bakalářské práce je vytvořit aplikaci pro ruční kontrolu automaticky zpracovávaných dat. Díky této aplikaci bude umožněno návštěvníkům portálu SuperLectures.com opravit textové přepisy slajdů nebo audia daného segmentu přednášky. Pokud se rozhodnou vypomoci, budou přesměrováni na Anotační portál, jak byla tato aplikace nazvána a bude jim přidělena elementární práce na opravu představující právě danou část přednášky. Nejdříve je provedena neformální analýza, která hledá možné postupy a prostředky pro řešení. Dále jsou požadavky specifikovány formálně a na základě nich je navržena vlastní aplikace. Podle návrhu a za použití vybraných prostředků a postupů je aplikace implementována. Na závěr proběhne její testování a zhodnocení výsledků.

Abstract

The objective of this bachelor thesis is to create application for hand corrections of automatically processed data. Thanks to this application it will be allowed to visitors of portal SuperLectures.com to correct text transcripts of slides or audio of the current segment of the lecture. If they decide to help, they will be redirected to the Annotation portal as this application was called and they will be assigned an elementary job to correct representing the given part of the lecture. At first informal analysis is performed, looking for possible tools and procedures for realization. After that requirements are formally specified and later the application design is based on them. Application is implemented according to the design with use of selected tools and procedures. In conclusion, the application is tested and test results are evaluated.

Klíčová slova

Web, webová aplikace, crowdsourcing, návrhové vzory, doménově řízený návrh, objektově-relační mapování, Model-View-Presenter, Entity-Repository-Mapper, PHP, Nette Framework, Doctrine2, HTML5, JavaScript, jQuery.

Keywords

Web, web application, crowdsourcing, design patterns, domain driven design, object-relational mapping, Model-View-Presenter, Entity-Repository-Mapper, PHP, Nette Framework, Doctrine2, HTML5, JavaScript, jQuery.

Citace

Zdeněk Coufal: Aplikace pro ruční kontrolu automaticky zpracovávaných dat, bakalářská práce, Brno, FIT VUT v Brně, 2012

Aplikace pro ruční kontrolu automaticky zpracovávaných dat

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Igora Szökeho, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Zdeněk Coufal
14. května 2012

Poděkování

Tímto bych chtěl poděkovat vedoucímu práce Ing. Igoru Szökemu, Ph.D. za vedení této práce a ochotu při konzultacích.

Obsah

Seznam obrázků

Kapitola 1

Úvod

V mnoha oblastech nám počítače usnadňují práci a život, ale někde je stále potřeba jim podat pomocnou ruku. V jedné z těchto oblastí působí i výzkumná skupina BUT Speech@FIT, kde se zabývají automatizovaným převodem řeči a obrazu do textové podoby. Otázkou je, jak ještě vylepšit kvalitu dosažených výsledků?

Zde přišla na svět myšlenka, že by se dalo využít síly kolektivního zpracování a nechat veřejnost přiložit ruku k dílu. Taková výpomoc se týká především oprav přepisů slajdů nebo audia. Ve výsledku lze shromáždit velké množství opravených dat od více uživatelů zároveň a po porovnání, vyhodnocení a sloučení jejich výstupů tak získat levné a přesto kvalitní opravy, které pomohou doplnit slovník a příští automatizované zpracování bude zase o kus dále.

Na základě zmíněných úvah byl stanoven cíl této práce, kterým bylo vytvoření aplikace, která umožní uživatelům ručně korigovat chyby v automaticky zpracovávaných datech. Tuto práci jsem zvolil, protože mě vždy zajímaly webové technologie a chtěl jsem se přiučit něčemu novému. Zároveň mi přišlo užitečné pracovat na něčem, co se bude v praxi opravdu používat a budou vidět reálné výsledky.

Pro realizaci tohoto nápadu jsem se rozhodl využít možností dnešních webových aplikací. Jsou snadno dostupné a lehce spravovatelné, během posledních let se staly rovnocenným soupeřem pro běžné desktopové aplikace.

V textu se dále zabývám neformální analýzou zadání viz kapitola 2, zde se snažím najít vhodné postupy a nástroje pro realizaci. Zejména takové, které umožní soustředit se na konkrétní problém a tím pádem odpadnou běžné rutinní záležitosti, které jsme často nuceni opakovaně řešit. Navazuje analýza formální viz kapitola 3, kde jsem rozebral konkrétní požadavky a získal tak podklady pro vlastní návrh viz kapitola 4. Jak jsem aplikaci uvedl do života popisuji v kapitole 5. V téže kapitole se zabývám testováním a zhodnocením dosažených výsledků.

Kapitola 2

Neformální analýza

Za základ vytvoření aplikace, se kterou bude zákazník spokojen, považuji kvalitní analýzu, proto jsem jí věnoval dostatek času. V úvodní studii se věnuji důvodům, které daly vzniknout této práci a požadavkům na aplikaci viz sekce 2.1.

Dále rozebírám, proč je vhodné koncipovat aplikaci jako webovou viz sekce 2.2, pokračuji výběrem vhodné architektury viz sekce 2.3 a nástrojů, kterými lze realizovat viz sekce 2.4.

2.1 Úvodní studie

Úvodní studie je zaměřena na vytyčení cílů a sepsání neformálního popisu pro vznikající aplikaci. Staví na konzultacích se zadavatelem, kterým je zároveň vedoucí této práce. V průběhu času se záměry měnily a tato zpráva tak obsahuje poslední stanovené.

Hlavním důvodem pro vznik této aplikace je potřeba oprav přepisů slajdů a audia pro portál *SuperLectures.com*. Tento portál prezentuje přednášky inovativním způsobem a to tak, že kromě vlastní přednášky má návštěvník k dispozici navíc textový přepis řeči a slajdy. Vše je synchronizováno a může se tak pohybovat v čase přednášky výběrem požadovaného textu z přepisu nebo konkrétního slajdu.

Někdy se může stát, že počítač při dolování dat nerozpozná daná slova správně, důvody mohou být třeba, špatná čitelnost nebo srozumitelnost materiálu. A v takovém případě musí chybná místa nalézt a opravit člověk. Protože dat je opravdu hodně a pár lidí by na tom dělalo celou věčnost, zapojení veřejnosti se jeví jako ideální. O tomto způsobu využití veřejnosti se můžete dočíst v sekci 2.5.

Cílovou skupinou jsou především návštěvníci portálu *SuperLectures.com*. Jestliže při přehrávání nějaké přednášky narazí na chybu a budou ochotni vypomoci, budou přesměrováni na tuto aplikaci, kterou jsme nazvali *Anotační portál*. Do budoucna by se mohlo jednat o autonomní portál, jehož funkčnost by mohla být rozšířena o další způsoby přidělování práce a další typy vykonávané práce.

2.1.1 Požadavky

Stěžejním požadavkem je vytvořit systém na principech crowdsourcingu viz sekce 2.5, který umožní uživatelům opravovat či kontrolovat přepisy slajdů a audia v podobě jednotlivých prací. Jedná se tedy o typ crowdsourcingu nazývaný „Microwork“[1].

[1] Uživatel řeší úkol, na jehož zvládnutí nemá počítač dostatek schopností.

Požadavky na způsob distribuování práce: (v rámci této práce je vyžadován první bod, zbytek jsou možná rozšíření funkcionality do budoucna)

- *Uživatel má možnost přistoupit k práci přes portál SuperLectures.com tak, že kdykoliv v průběhu přednášky může zvolit „opravit přednášku“, což ho přesměruje na korespondující část přednášky zadanou v aplikaci jako elementární práce. Jsou to práce menšího rozsahu, většinou o velikosti maximálně pár desítek položek (slajdů, segmentů audia) a není za ně žádná odměna.* Jelikož jde o práce vykonávané anonymně a není za ně žádná odměna, není počet uživatelů na jednu práci omezen.
- Uživateli bude vytvořen účet v systému administrátorem, bude mu přidělena práce a následně tato práce bude nabízena na větších portálech fungujících na principech crowdsourcingu jako je *oDesk.com, Elance.com, Guru.com.* Tomu, kdo o tuto práci projeví zájem, budou emailem zaslány přihlašovací údaje do systému. Až práci dokončí, spolupráce končí, jde o jednorázovou záležitost většinou mířenou do Indie. Počet uživatelů na jednu práci je omezen.
- Uživatel se sám zaregistruje a bude mít možnost si sám vybírat práce nabízené v systému, jedná se o spolupráci dlouhodobou. Stejně jako v předchozím případě i zde je počet přiřazených uživatelů na jednu práci omezen. Není vyloučena možnost práce vykonávat dobrovolně, ale bez odměny.

Požadavky na administrátorské role v systému:

- Administrátor má možnost spravovat uživatelské účty v systému. V rámci hierarchie by měly být dostupné dva typy administrátorských účtů. První má pravomoce ke všemu a přehled o všem, druhý má možnost dohlížet pouze na práce a uživatele v jeho kompetenci.

Požadavky na správu prací:

- U každé práce lze rozhodnout, zda bude nabídnuta registrovaným uživatelům volně k výběru, nebo zda přijímá dobrovolné (neplacené) řešitele[2]. Dále jestli má nějaký limit počtu placených řešitelů a jaká bude odměna za kvalitně odvedenou práci v případě, že se jedná o placeného řešitele. Také lze stanovit deadline, do kterého musí přiřazení řešitelé dokončit práci, jinak bude automaticky uzamčena.
- Možnost již rozdělanou, ale nedokončenou práci, převést na jiného řešitele.
- Sloučit dobře ohodnocené práce a vytvořit tak jednu opravdu kvalitní práci, kterou lze brát jako výsledek.
- Přidávání i získávání výsledků všech prací bude probíhat pomocí XML souborů, přenos mezi oběma portály bude probíhat manuálně.
- Systém musí umožnit hromadné vložení prací, může se jednat o stovky až tisíce elementárních prací.
- Administrátor má možnost vkládat do systému práce, sledovat průběh jejich zpracování a vyhodnocovat je.

[2]Těmi mohou být jak anonymní, tak registrovaní uživatelé, kteří se rozhodli přispět k výsledku, ale bez odměny.

Požadavky na vykonávání prací:

- Protože se nelze spolehnout na to, že každý uživatel odvede kvalitní práci, je vhodné nechat více uživatelů pracovat na stejné práci.
- Řešitel se může k jednou rozdělané práci vrátit později, je tedy potřeba ukládat průběžně stav.
- Činnost řešitele bude zaznamenávána, aby bylo možné odfiltrovat ty nepracující.

Požadavky na vyhodnocování prací:

- Administrátor má možnost hodnotit práci již v průběhu jejího zpracování řešitelem a dělat závěry.
- Systém musí automaticky tyto práce porovnávat a adekvátně je ohodnotit. Pokud není k automatickému hodnocení dostatečný počet prací, administrátor provede vyhodnocení ručně.

Požadavky na chod systému:

- Systém bude hlídat překročení deadlinů a automaticky zmrazí rozdělané práce, pokud překročí hranici jisté doby nečinnosti[3].

Do budoucna by mělo být možné aplikaci rozšířit o nějaký další typ vykonávané práce. Tento požadavek klade přímé nároky na modularitu a rozšiřitelnost.

2.2 Web jako platforma

Webová aplikace, tak jak ji známe dnes, musela urazit pořádný kus cesty, než získala svoje současné postavení. Samotná myšlenka není zase až takovou novinkou. Přišla s ní firma Netscape již koncem minulého století, ale v té době chybějící technologie a nedostatečná vyspělost přenosu dat neumožnila konkurovat desktopovým aplikacím [3].

Od roku 2004, kdy nastává éra Webu 2.0, se toho hodně změnilo. Je to období kdy se objevují různé komunitní servery, systémy pro sdílení, blogy a podobně. Web ožívá v pravém slova smyslu a stává se tak živnou půdou pro webové aplikace.

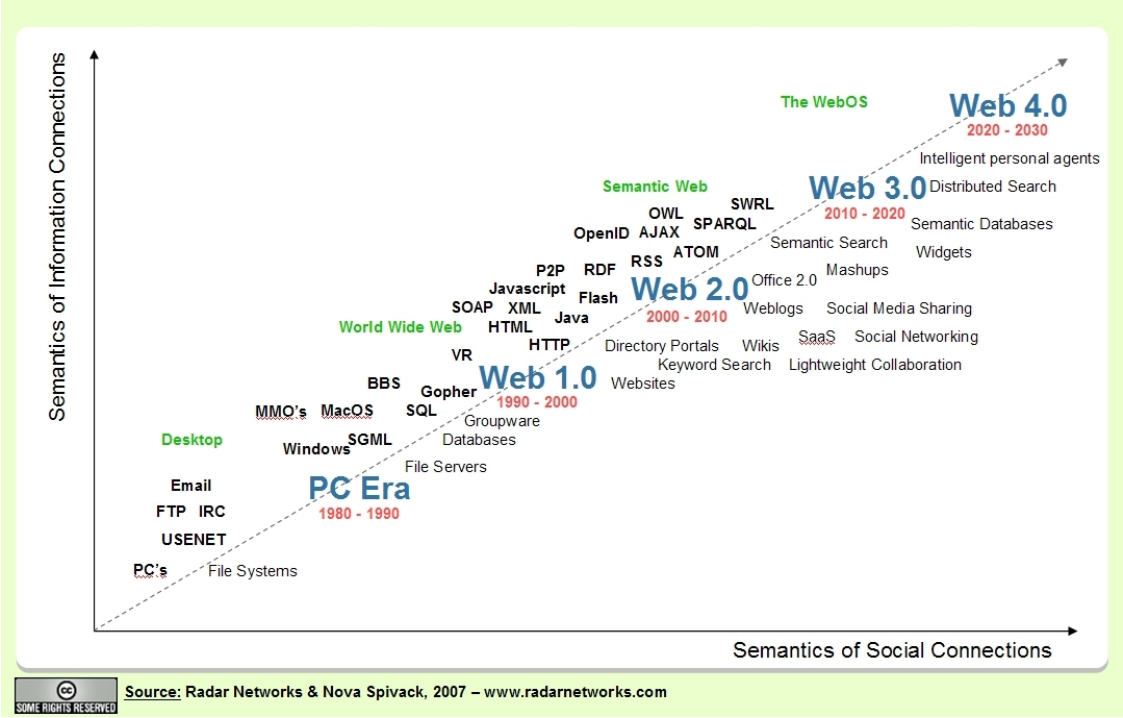
Našel jsem pěkné rozdělení vývojových fází webu na dekády, se kterým přišel Nova Spick, významná osobnost v oblasti webu. Nejlépe myšlenku vystihuje časová osa, kterou navrhl, viz obrázek 2.1. Podle této časové osy by jsme se dnes, tedy v roce 2012, měli právě nacházet v etapě s označením Web 3.0. Na schématu můžete vidět také technologie, které přicházely a budou přicházet postupně v čase.

Jak je vidět, pojem webové aplikace se dostává do povědomí lidí až v posledních letech. Jak ji tedy definovat a rozlišit od typických webových stránek?

> *Webová aplikace je jakákoliv aplikace, která používá webový prohlížeč jako klienta. Složitost aplikace jde od obyčejné knihy návštěv až po komplexní řešení typu textový či tabulkový procesor, CAD systémy[4] nebo nástroje pro editaci videa.*[13]

[3] To částečně zajistí odfiltrování těch uživatelů, kteří jenom nakouknou a jejich práce zůstane v systému rozpracovaná a nedokončená. Zmražením bude vyjmuta z automatického hodnocení.

[4] CAD neboli Computer-Aided Design jsou nástroje k vytváření a dokumentování návrhů v mnoha oblastech, např. strojírenství nebo stavebnictví, za pomoci počítače.



Obrázek 2.1: Schéma vývojových etap webu podle Novy Spicka. Převzato z [15]

Jaké výhody a nevýhody přináší použití webové aplikace oproti desktopovým?[12]

- Odpadá potřeba provádět hromadné instalace v rozsáhlých organizacích, stačí kompatibilní webový prohlížeč.
- Aktualizace probíhají centralizovaně přímo na serveru.
- Jsou multiplatformní. Vývojářům odpadá starost vyvíjet aplikaci pro konkrétní typ počítače či operačního systému.
- Řeší problémy se synchronizací a bezpečností, centralizovaná správa dat je jistě výhodou.
- Většina výpočetních operací probíhá převážně na straně serveru, ale nelze tvrdit, že prohlížeč je pouze nástroj pro zobrazení výstupu.
- S úrovní dnešních technologií a hlavně příchodem *HTML5 (HyperText Markup Language)* lze vytvářet velice sofistikované a interaktivní aplikace.

I přes výčet benefitů webové aplikace se najdou některé vlastnosti, které nelze označit za nedostatek, nýbrž přirozený proces vývoje.

- Podmínkou je dostupné připojení k Internetu, ale to už se dnes stává samozřejmostí. Také se objevují řešení pro offline zpřístupnění těchto aplikací.
- Společnosti provozující tyto aplikace mohou sledovat akce uživatele a všechno, co dělá. Otázkou je, zda to nehraničí s narušováním soukromí.

- Nutnost znovunačítání stránky (aplikace) při každém požadavku. Dnes je tento stav řešen částečně technologií *AJAX (Asynchronous XML and JavaScript)* a pomalu se objevují i takové nástroje, které simulují nebo fungují na stejném principu jako běh desktopové aplikace [5].

Webové aplikace získávají na popularitě a myslím si, že skrývají velký potenciál. Ale jak obstojí v konkurenci běžným desktopovým aplikacím, to ukáže až čas.

2.3 Architektury ve webových aplikacích

Jak se stávaly webové aplikace složitějšími a rozsáhlejšími, dostává se pojem architektura a s tím spojené využití architektonických a návrhových vzorů i do této oblasti. Objevují se principy a postupy, které usnadňují návrh těchto „enterprise“ aplikací viz podsekce 2.3.1. Protože výše zmíněných principů využívám i v této práci, dále se podrobněji zabývám problematikou těch, které jsem se rozhodl použít.

Architektonické vzory jsou tedy způsobem, jak rozdělit aplikaci do více částí, které tvoří logické celky a způsobem popisujícím jejich vzájemné interakce[10]. Zjednoduššují pozdější úpravy, které se většinou vztahují na konkrétní vrstvu.

Návrhové vzory se zase mohou vyskytovat jako součásti výše zmíněných architektonických vzorů, kde zastávájí strukturu konkrétní vrstvy aplikace.

O použití těchto vzorů a návrhu komplexních aplikací se můžete dočíst v knize Martina Fowlera[6], který se touto oblastí zabývá již řadu let a stal se uznávaným odborníkem na toto téma. Odtud jsem také čerpal vysvětlení mnoha pojmů.

2.3.1 Domain Driven Design

Pojem *DDD (Domain Driven Design)* tzv. doménově řízený návrh je dílem Erica Evanse[5]. Je to přístup, jak řešit komplexní softwarové problémy, které jsou pak snáze pochopitelnější a spravovatelnější. Nabízí praktiky, které usnadňují rozhodování ve fázi návrhu a tím urychlují proces vývoje. Snahou je soustředit vývoj aplikace na doménu vyvíjeného systému, která se postupem času vyvíjí [6].

Vhodným doplněním tohoto návrhu je použití architektonických a návrhových vzorů. Používaným architektonickým vzorem při návrhu webových aplikací může být *MVC (Model-View-Controller)* a jeho deriváty jako *MVP (Model-View-Presenter)* viz podsekce 2.3.2.

MVC/MVP na úrovni datové vrstvy skvěle doplňuje *ORM (Object-Relational Mapping)* viz podsekce 2.3.3 tzv. objektově relační mapování, které představuje pro nás lidi čitelnější a průhlednější způsob práce s relačním datovým modelem za použití objektově orientovaného paradigmatu. A to přímo koresponduje s doménově řízeným návrhem.

[5] Tím mám na mysli, že aplikace opravdu skutečně po celou dobu činnosti běží a je ukončena až na povel uživatele, kdežto u typické webové aplikace co jedna akce uživatele, to požadavek na webový server a spuštění interpretace skriptů. Částečně lze lze tuto činnost optimalizovat kešováním již jednou zpracovaných požadavků.

[6] Doména je realizací objektů a závislostí skutečného světa.

2.3.2 Model-View-Controller

Návrh aplikace jsem postavil na vzoru MVP, ale protože vychází právě z MVC, je pro jeho pochopení potřeba začít právě u MVC.

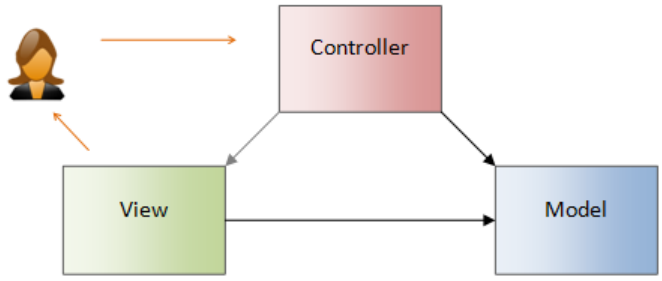
Historie tohoto vzoru sahá až do 70.let minulého století, poprvé byl popsán v roce 1979 Trygvem Reenskaugem. A použit byl poprvé v jazyce Smalltalk, vyvíjeném v Xerox research labs[9]. V této době neprobíhalo žádné dělení logiky aplikace a uživatelského rozhraní, tato přílišná provázanost vedla ke komplikacím při změnách, zvyšovala složitost a byla problémem při testování. Ukázalo se, že řešením je oddělit kód aplikační logiky od uživatelského rozhraní.

Hned ze začátku bych podotknul, že je potřeba si uvědomit rozdíl mezi MVC jako přístupem ke tvorbě aplikací, tedy v tomto smyslu je obecnou architekturou. A pak jednotlivými variantami v podobě architektonických vzorů.

Základní dělení[2]:

- **Model** – představuje data a business logiku aplikace, někdy též nazývanou jako doménová logika. Pokud je o tuto logiku ochuzen a tvoří jej pouze datové objekty, je nazýván jako Anémický model[6].
- **View** – představuje uživatelské rozhraní.
- **Controller** – má na starosti tok událostí v aplikaci a obecně aplikační logiku.

Třídám reprezentujícím jednotlivé komponenty se pak říká **triády**. V rámci klasické třívrstvé architektury[7] se MVC řadí do prezentační vrstvy. Schéma MVC můžete vidět na obrázku 2.2.



Obrázek 2.2: Schéma architektonického vzoru Model-View-Controller. Převzato z [2].

Schéma vyjadřuje tok následujících událostí v aplikaci:

1. Uživatel interaguje s uživatelským rozhraním.
2. Akce uživatele je zachycena Controllerem.

[7] Třívrstvá architektura je nejznámějším případem vícevrstvé architektury. Často je využívána právě u webových aplikací. Obsahuje prezentační vrstvu, která zobrazuje informace uživateli. Aplikační vrstvu, ta tvoří jádro aplikace, její logiku, funkce, výpočty a zpracování dat, je na straně webového serveru. A poslední je vrstva datová, kterou nejčastěji tvoří databáze.

3. Controller rozhodne o reakci na akci (změna hodnot v Modelu či rozhodne, které View zobrazit).

4. View promítne změny uživateli.

Jak konkrétně funguje v prostředí webu?

- **Model** je identický s Modelem v desktopových aplikacích, obsahuje data a doménovou logiku.

- **View** bývá někdy mylně považováno u webové aplikace za HTML, které je ve výsledku zobrazeno, ale ve skutečnosti je to serverový kód, který se o generování HTML stará, takže PHP, Java a podobně. Výstupem nemusí to být nutně HTML, ale klidně XML nebo JSON a další formáty.

- **Controller** se skládá ze dvou hlavních částí, první je Front Controller, který zachytává HTTP požadavky a deleguje je konkrétním Controllerům, ty tvoří část druhou. Konkrétní Controller má na starosti zpracovat data původně obsažena v HTTP požadavku, třeba je uložit do Modelu a následně zavolat patřičné View, které už se postará o vykreslení HTML.

Pointou tohoto vzoru je, že uživatelské vstupy má na starosti Controller. A jeho použití má význam právě u těch webových aplikací, kde je vstup a výstup oddělen. Takže pokud vstupem je URL, výstupem je třeba HTML.

Ale objevily se i webové aplikace widgetového typu, kde View reprezentující uživatelské rozhraní dokáže zpracovat jak vstup, tak i výstup a zde nachází uplatnění další vzor z rodiny MVC a tím je *MVP (Model-View-Presenter)*.

Principiálně se význam MVC v podání webových aplikací tolik neliší od původního návrhu MVC na desktopu.

Model-View-Presenter Zásadní změnou oproti MVC je, že vstup i výstup má u tohoto vzoru na starosti View.

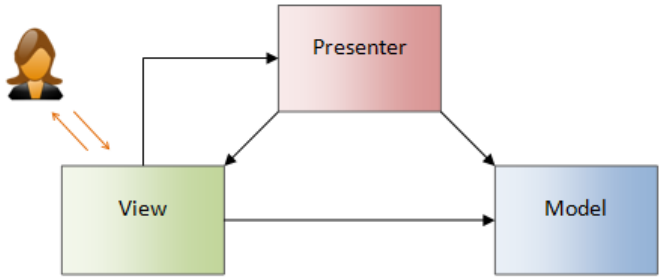
A jak to ovlivní jednotlivé komponenty?

- **Model** se nezměnil.

- **View** oproti MVC zpracovává uživatelský vstup, při reakci uživatele na nějakou akci typicky zavolá metodu Presenteru, jde tedy o delegaci akcí. Aplikační logika sem rozhodně nepatří.

- **Presenter** stará se o aplikační a prezentační logiku, přímo manipuluje s Modelem a následně View a nebo je View o změnách informováno pomocí notifikací, třeba použitím návrhového vzoru Observer.

Výsledné změny lépe vyjádří schéma na obrázku 2.3, ze kterého vyplývá, že veškerá komunikace s uživatelem probíhá skrze View.



Obrázek 2.3: Schéma architektonického vzoru Model-View-Presenter. Převzato z [2].

Shrnu-li hlavní myšlenku použití MVC a MVP[2]:

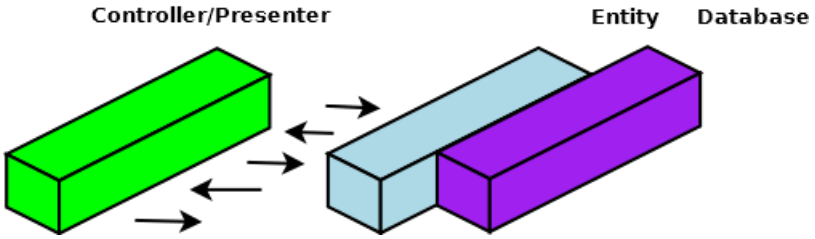
- Oddělený vstup a výstup – lze aplikovat vzor MVC a vstup zpracovává Controller.
- Platforma s konceptem ovládacích prvků, tj. prvků, které dokáží obsloužit jak vstup, tak i výstup – lze aplikovat vzor MVP, vstup zpracovává View a úloha Presenteru se mění.

2.3.3 Object-Relational Mapping

Umožňuje konvertovat data mezi nekompatibilními typy systémů za použití objektově orientovaného programování. To vytváří vrstvu „virtuální objektové databáze“, se kterou se ve zbývajících vrstvách aplikace lépe pracuje. Osobně považuji objektově-relační mapování za vhodnou součást doménově řízeného návrhu při modelování datové vrstvy.

Pro realizaci ORM jsou obvykle využívány návrhové vzory *Active Record* a *Entity-Repository-Mapper*. V dalším textu uvádím jejich porovnání[16].

Active Record Nejdříve tu byl starý dobrý Active Record, který mapoval 1:1 strukturu dat z databáze na třídy. Grafickou reprezentaci modelu rozděleného do vrstev na základě tohoto vzoru můžete vidět na obrázku 2.4



Obrázek 2.4: Schéma představující strukturu modelu při použití návrhového vzoru Active Record.

Charakteristika vzoru:

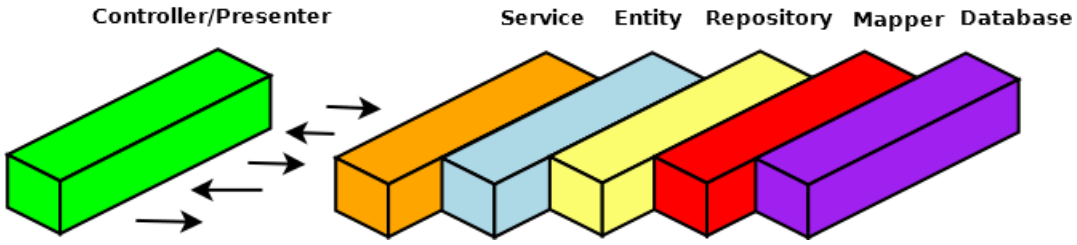
- Každá tabulka databáze odpovídá jedné třídě.
- Řádek tabulky odpovídá instanci třídy.

- Všechny metody zahrnující doménové operace i vlastní persistenci dat jsou v jedné třídě.

Ačkoliv je Active Record jednoduchý a tím poměrně jasný, je zde několik překážek, na které se dá narazit.

- *Jedna třída, jedna tabulka* – to může být problémem v případě, kdy entita překrývá více tabulek.
- *Silná vazba na strukturu databáze* – problém při změně typu úložiště, obzvláště pokud se jedná o typ, který nemá s relačními databázemi nic společného.
- Použití *více různých typů uložišť* v rámci jedné entity nepřipadá v úvahu.
- *Rozšiřitelnost* – prakticky nulová, dá se využít dědičnosti, ale i tak je to zásah do třídy entity, což není dobré.
- Porušuje princip *Single Responsibility*[8] tím, že je jedna třída zodpovědná za všechno.
- *Obtížně testovatelné.*
- *Není to ten model, který v rámci doménově řízeného návrhu chceme.*

Entity-Repository-Mapper: Protože se Active Record ukázal v některých případech jako nedostačující, vznikl jeho zobecněním a rozšířením o nové vrstvy návrhový vzor *Entity-Repository-Mapper*. Stejně jako v předchozím případě jeho grafickou reprezentaci můžete vidět na obrázku 2.5



Obrázek 2.5: Schéma představující strukturu modelu při použití návrhového vzoru Entity-Repository-Mapper.

Popis jednotlivých vrstev:

- **Entity** – odráží strukturu jedné nebo i více tabulek, představuje logický celek v rámci aplikační domény. Definuje i doménová pravidla. Může být v kompozičním vztahu s jinou entitou, tj. být její součástí.
- **Repository** – tato vrstva převzala zodpovědnost za ukládání, načítání a mazání entit, k provedení těchto operací využívá funkce konkrétního Mapperu.

[8] Třída by měla mít zodpovědnost za jednu věc, kterou zapouzdřuje.

- **Mapper** – pracuje s konkrétním úložištěm, tím může být třeba databáze, memcache, souborový systém nebo nějaká jiná vzdálená služba. Je zodpovědný za veškerou obsluhu jednoho typu úložiště.
- **Service** – vrstva sdružuje více operací nad jednou nebo více entitami, které tvoří logický celek.

Výhodou tohoto vzoru je bezpochyb možnost bezbolestně změnit typ úložiště bez markantních zásahů do dalších vrstev aplikace. Například v případě užití vzoru MVP je v Presenteru využito pro práci s entitami pouze repozitářů a pokud by byla nutná změna úložiště, stačí vyměnit Mapper, který repozitář využívá.

Objektově-relační mapování využívá kromě těchto vzorů ještě vzor *Unit of Work* viz [6], ten uchovává všechny provedené změny na entitách a aby se předešlo elementárním transakcím při každé operaci, provede aktualizaci databáze až když je jich více. Na programátorovi je, aby odhadl, kdy už je vhodné promítnout změny do databáze, v praxi totiž musí počítat s konkurentním přístupem k datům a je tak potřeba řešit navíc platnost těchto dat.

2.4 Výběr nástrojů

Základním kritériem pro výběr těch správných nástrojů pro mě bylo, abych se mohl soustředit pouze na řešení zadaného problému a neztrácel zbytečně čas na rutinních činnostech. Nakonec spousta webových technologií už má svoji rannou fázi za sebou a snaží se využívat elegantní a čistá řešení. Taková řešení nabízejí frameworky nebo celé platformy.

Zpracování na straně serveru Pro zpracování na straně serveru jsem zvolil jako základ skriptovací jazyk PHP, na kterém staví celá škála frameworků. Požadoval jsem, aby framework pracoval s verzí jazyka 5.3, ta totiž podporuje jmenné prostory a garbage collector. Dále, aby stavěl na architektuře MVC, měl k dispozici systém šablonování a umožnil využít vlastní datovou vrstvu třeba jako ORM. Také aby nevyužíval až příliš generování tříd a podobně.

- **Symfony** - podporuje PHP5, staví na MVC, nemá systém šablonování, umožňuje ORM, dosti využívá generování.
- **Zend** - splňuje všechny požadavky, využívá MVC architektury, má spoustu nachystaných pluginů, view helperů. Je to velmi propracovaný a přitom volně vázaný systém.
- **Nette** - není postaven přímo na MVC, ale na MVP, což není problém. Využívá hierarchie komponent a má vlastní systém šablonování (založený na regulárních výrazech a makro generátorech).

Nakonec jsem zvolil *Nette Framework*, které je přesně tou vrstvou abstrakce, kterou vyžaduji. Dal jsem mu přednost před Zend Frameworkem hlavně kvůli systému komponent, který mi přijde velice praktický. Také kvůli vlastnímu systému šablon*Latte*, který je přehledný a dobře se v něm pracuje. Dále mé rozhodnutí pro Nette utvrdila aktivní komunita českých PHP vývojářů a relativně strmá křivka rychlosti učení.

Zároveň s Nette jsem se rozhodl využít pro realizaci modelu ORM framework *Doctrine2*. Členění datové vrstvy na jednotlivé dílčí podvrstvy v rámci vzoru Entity-Repository-Mapper usnadní případnou změnu úložiště a navíc se zde dá pěkně využít doménově řízeného návrhu.

Zpracování na straně klienta U typu práce kontrola, oprava přepisů audia jsem hledal možnosti jak postavit vlastní přehrávač. Vybíral jsem mezi Flash a HTML5, nakonec jsem zvolil *HTML5*, protože umožňuje manipulovat s přehrávačem skrze JavaScript a také kvůli dalším novinkám, které tato nová specifikace jazyka přináší.

Dále klasicky JavaScript a na něm postavený framework *jQuery*, který smazává drobné implementační rozdíly napříč prohlížeči. Pro nastylování výsledného vzhledu klasické CSS.

Webový server Jako webový server poslouží často používaný Apache server.

Persistence dat Pro persitenci dat jsem vybral databázový relační server *MySQL*, který je v jisté verzi volně dostupný a od verze 5.0 s jeho funkčností může konkurovat i takovým velikánům jako je Oracle. Další výhodou je, že ho využívá mnoho webových hostingů.

2.5 Crowdsourcing

Crowdsourcing je distribuovaný problém řešící a produkční model. V běžném znění se jedná o otevřenost vůči řešením poskytnutých neznámou skupinou řešitelů. Tací řešitelé odevzdávají jejich verze řešení, jejichž vlastníkem se stává zadavatel úkolu. Řešitelé, přispívající ke konečnému řešení, bývají odměněni buď peněžně, uznáním nebo intelektuálním uspokojením. Řešiteli mohou být amatéři či dobrovolníci pracující v jejich volném čase nebo experti či malé podniky, které jsou pro zadavatele neznámé [7].

Zadavatelé, kteří využívají služeb v podobě crowdsourcingu, jsou motivování jeho četnými výhodami. Zisk velkého množství řešení nebo informací vzhledem k relativně nízkým nákladům. Oproti tomu uživatelé přispívající svými řešeními mohou být motivování vnitřně, kdy se jedná o využití času či sociální kontakt a nebo může být podnět z vnějšku v pobobě finanční odměny.

Rozmazané hranice tohoto vysvětlení crowdsourcingu způsobily, že mnoho aktivit postavených na kolaboraci, bývá považováno za crowdsourcing i když jím nejsou. Taková proliferace definic autory vzhledem k jejich poli působnosti vedla ke ztrátě globálního pohledu na vysvětlení, co to vlastně ten crowdsourcing je.

Tento stav změnili až pánové Estellés a González, kteří navrhli novou jednotnou definici sjednocující 40 dosavadních.

> *Crowdsourcing je typ participativní online aktivity, ve které jednotlivec, instituce, nezisková organizace nebo společnost vyzývá skupiny jednotlivců různých zkušeností a počtu, k flexibilnímu dobrovolnému provedení úkolu. Provádění úkolů různých složitostí a modularity, ve kterých se veřejnost účastní přispíváním jejich odvedenou prací, penězi, zkušenostmi a znalostmi, vždy zahrnující vzájemné výhody. Uživatelé obdrží zadostiučinění dané typem jejich potřeby v podobě ekonomické, společenského uznání, sebedůvěry či zvýšením individuálních schopností, zatímco zadavatel obdrží a využije ke svému prospěchu to, čím uživatel přispěl k provedení úkolu, čehož forma závisí na typu prováděného úkolu.*[4]

Definice se může zdát vyčerpávající, ale postihuje všechny níže uvedené kategorie viz podsekce 2.5.2.

2.5.1 Historie

Poprvé pojem definoval Jeff Howe v roce 2006[7], ale existenci principu lze na mnoha příkladech vypozorovat již v minulosti.

Byl takto vytvořen známý Oxfordský anglický slovník, kdy byly vyzváni obyvatelé Spojeného království, aby dobrovolně postihli všechna použití anglických slov i s příklady. Bylo nastřádáno 6 milionů přispěvků za dobu 70 let.

Existoval také v podobě různých soutěží, kdy se hledalo nějaké řešení. Francouzská vláda takto v minulosti pořádala několik soutěží, většinou pro chudé Francouze, kteří odvedli vskutku dobrou práci. Příkladem budiž Leblancův proces, kde šlo o oddělení soli a alkálií nebo Fourneyronova turbína, která byla první hydraulickou turbínou.

Jak můžeme vidět, i když crowdsourcing jako pojem není znám příliš dlouho, jeho principů bylo využíváno již před několika stoletími.

2.5.2 Dnešní podoba

Dnes se s ním můžeme setkat převážně v prostředí webu, který od etapy Web 2.0 již umožňuje dosažení těchto podnikatelských záměrů. Ukázalo se, že lidé jsou více otevřeni webovým projektům, protože jsou zde jistým způsobem anonymní.

Jsou dvě cesty, kterými lze jít. Máme tu *explicitní crowdsourcing*, uživatelé společně pracují na vyhodnocení, sdílení a vytváření různě specifických úkolů a pak je tu *implicitní crowdsourcing*, uživatel řeší nevědomky problém jako vedlejší efekt nějaké jeho činnosti.

Otec tohoto pojmu Jeff Howe naznačil také společné kategorie, které lze efektivně užívat v komerčním prostředí [8].

- *Crowdvoting* je sběr názorů a posudků velké skupiny lidí na určité téma. V případě marketingu tak lze zapojit spotřebitele do procesu již ve fázi vývoje produktu tzv. „co-creation“[14].
- *Wisdom of the crowd* vychází z myšlenky „více hlav více ví“, shromážděním obrovského množství informací a jejich agregací lze získat kompletní a přesný pohled na danou věc. Myšlenka kolektivní inteligence se ukázala zvláště efektivní na webu, protože lidé mohou přispívat v realném čase a z celého světa.
- *Crowdfunding* je proces financování projektů množstvím lidí přispívajících malou částkou za účelem dosáhnout určitého peněžního cíle. Tito lidé jsou často rekrutování ze sociálních sítí při nákupech, půjčkách, darech a podobně.
- *Microwork* je platforma, kde uživatelé vykonávají malé úkoly, na jejichž zvládnutí nemají počítače dostatek schopností, většinou za malou finanční odměnu.
- *Inducement prize contests* jako kolektivní tvorba myšlenek a následná diskuze, platforma postavená na Internetu. Takto otevřené platformy jsou efektivní cestou, jak crowdsourcovat myšlenky a nápady lidí do vývoje a výzkumu.
- *Implicit crowdsourcing* je zároveň speciálním typem.

Zeptáme-li se, co motivuje obě strany k využití tohoto systému, dostáváme následující odpovědi: Pro zadavatele je výhodné a efektivní croudsourcování spíše menších úkolů, u složitějších se mohou střetnout s nějakými komplikacemi viz podsekce 2.5.4. Úkoly bývají z oblasti vědy, výroby, biotechnologií a medicíny.

- Levná pracovní síla a informace. Zda je to etické, je předmětem mnoha diskuzí.
- Přístup k mnoha talentům, kteří jsou soustředěni mimo organizaci.
- Zvládnutí úkolů, které jsou pro interní řešení příliš náročné.
- Dosažení lepších výsledků.

Z pohledu řešitelů je významným aspektem otázka motivace, ta může být vnitřní nebo vnější.

Vnitřní motivace může být založena na zábavě při řešení úkolu, to zahrnuje řešení různě obtížných autonomních úkolů, zpětnou vazbu a příjemně strávený čas. Dále kontakt s komunitou, obecně sociální kontakt a hledání svého místa.

Vnější motivací může být okamžitá platba za odvedenou práci. Pak opožděná platba v podobě benefitů, které mohou být do budoucna výhodou, např. trénování vlastních schopností nebo upozornění na sebe u potencionálních zaměstnavatelů. Stejně tak pocit, že bylo vykonáno něco pro lidi, tzv. altruistické cítění či naopak prestiž nebo status.

V dnešní době se crowdsourcing ukázal jako systém efektivní, s ním narůstá důležitost komunity. Do budoucna bude využíván ve stále více oblastech jako je novinařina, programování, marketing.

2.5.3 Existující řešení

Příkladem *implicitního crowdsourcingu* je *reCaptcha*, které slouží nejen k prokázání, že se jedná o lidskou bytost, ale zároveň uživatel pomáhá dešifrovat části starých knih, které se nepodařilo digitalizovat pro webové použití. Patří sem i ESP hra, ve které se uživatel snaží rozpoznávat místa na obrázcích, výsledky jsou použity k otagování Google images.

Existujícími řešeními *crowdvotingu* je portál *Threadless.com*, prodávající trička s motivy navrženými a odhlasovanými vlastní komunitou spotřebitelů. Dále portál *iStockPhoto*, fungující na stejném principu, ale produktem jsou fotografie.

Portály jako americký *Mechanical Turk* nebo čínské *Witkeys* zase fungují na principu *microwork*.

2.5.4 Nástrahy

Jako každý idealizovaný systém i crowdsourcing skrývá mnohé nástrahy a pasti, se kterými je dobré počítat. Tím lze předejít nečekaným výsledkům a dalším nepříjemným situacím.

Zadavatel by měl brát v potaz:

- Někdo může záměrně kazit práci a tím snižovat hodnotu výsledku. Důvodem bude pravděpodobně nutkání řešitele vypracovat úkol co nejrychleji spíše než kvalitně. To vede k potřebě povolat více řešitelů na stejný úkol a to jsou bohužel výdaje navíc.
- Řešitelé nejsou vybírání zcela náhodně napříč populací, ale díky nízkým odměnám za odvedenou práci, se jimi stávají spíše uživatelé z vývojových zemí.
- Zvýšená pravděpodobnost, že projekt selže kvůli špatné motivaci uživatelů a nebo nízkému počtu řešitelů.
- Z etických důvodů se mohou jevit nízké platby za odvedenou práci jako nedostatečné.

Řešitel je vystaven:

- Žádné písemné smlouvy, dohody o provedení práce.
- Zadavatel má poslední slovo v tom, zda je vykonaná práce přijatelná či nikoliv.
- Ztížené podmínky spolupráce s ostatními řešiteli, zvláště, jedná-li se o soutěživá zadání práce.
- Nízké mzdy.

Kapitola 3

Formální analýza

Na základě neformální analýzy jsem definoval uživatelské role a jejich případy užití viz příloha 3.1. Pak jsem namodeloval konceptuální schéma databáze pomocí *ER (Entity-Relationship)* diagramu viz příloha 3.3. Také jsem stanovil systém prací viz sekce 3.2.

3.1 Uživatelé a jejich případy užití

V systému se budou vyskytovat dvě skupiny uživatelů. Skupina administrátorů, starající se o chod systému a pak skupina běžných uživatelů vykonávajících práce.

- **Anonymní uživatel – nepřihlášený uživatel – host**, má možnost vykonávat pouze práce označené pro anonymní zpracování a zaregistrovat se do systému.
- **Jednorázový uživatel** – zaregistrovaný administrátorem, má možnost prohlížet práce, které mu byly přiděleny a vykonávat je.
- **Regulární uživatel** – běžně zaregistrovaný, má možnost vybírat si z nabízených prací, prohlížet přidělené práce a vykonávat je.
- **Administrátor přes práce** – může vkládat do systému práce, vytvářet jednorázové účty a přidělovat jim práce. Má přístup pouze k pracím a uživatelům v jeho kompetenci, s těmi může nakládat stejně jako hlavní administrátor.
- **Hlavní administrátor** – může v systému cokoliv, má přehled o všech pracích a uživatelích.

Diagram případů užití pro běžné uživatelské role viz příloha A.1 a diagram případů užití pro administrátorské role viz příloha A.2. Všechny role jsou ve vztahu dědičnosti, takže každý případ užití bude rozebrán pouze u role, kde se poprvé vyskytuje nebo pokud je pozměněn.

Anonymní uživatel

- **Manipulace s prací** – Uživatel nemá přímý přístup ke konkrétním pracím v systému, pokud na ně není odkázán z přednášky na portálu SuperLectures.com. Pokud tak učiní, je přesměrován na portál Anotací na práci související s aktuálním úsekem

přednášky. Práci může přerušit a pokračovat později, dokončenou práci odevzdává[1]. Jedná se o práce umožňující anonymní dobrovolné zpracování.

- **Registrace** – Může se zaregistrovat jako regulární uživatel, tím budou všechny jeho doposud anonymně vykonané práce navázány na jeho nově vytvořený účet.

Jednorázový uživatel

- **Manipulace s prací** – Může přistupovat k pracím přímo skrz portál Anotace, jedná se o natvrdo přiřazené práce.
- **Výpis přiřazených prací** – Uživatel má možnost vypsat všechny práce, které byly jeho účtu přiřazeny.
- **Registrace** – Může se zaregistrovat jako regulární uživatel.
- **Přihlášení do, Odhlášení ze systému** – Při přihlášení má možnost se rozhodnout, jestli chce zůstat v systému trvale přihlášen i po jeho odchodu. Zároveň s přihlášením budou navázány na jeho účet všechny práce, které případně udělal anonymně.

Regulární uživatel

- **Manipulace s prací** – Může přistupovat k pracím přímo skrz portál Anotace, může jít o práce, které si sám vybral.
- **Výpis prací** – Krom toho, že může vypsat všechny jemu přiřazené práce, má možnost prohlížet a vybírat si z volně dostupných nabízených prací.

Administrátor přes práce Akce prováděné administrátorem tohoto typu jsou pouze v rámci jeho kompetence tzn. má přístup pouze k uživatelům a pracím, které on sám vytvořil.

- **Správa uživatelů** – Může vytvářet účty jednorázových uživatelů, upravovat, mazat a u každého účtu má možnost vypsat jemu přiřazené práce a dokončené práce.
- **Správa prací** – Do systému může vkládat práce bez omezení (i hromadně), přiřadit jim řešitele, měnit jejich stav a smazat je. U každé práce má možnost prohlížet přiřazené řešitele, prohlížet jejich práce během zpracování, v případě potřeby práce smazat nebo zmrazit, předat práci jinému řešiteli, manuálně vyhodnocovat, uzavřít[2] a stáhnout výsledek[3].

Hlavní administrátor Hlavní administrátor nemá žádná omezení, jeho hlavním účelem je dohlížet na dění v celém systému.

- **Správa uživatelů** – Může vytvářet účty všech typů, sledovat u nich přiřazené práce, nabízené práce, dokončené práce.

[1] Odevzdat k vyhodnocení může pouze dokončenou práci, nesmí chybět žádné nezpracované položky.

[2] Zahrnuje vytvoření záznamu o provedené práci a potvrzení či navrhnutí odměny za řešení.

[3] Stáhnout může výsledek konkrétní práce nebo vybrat více prací pravděpodobně s dobrým hodnocením a sloučit jejich výstupy.

Čas V tomto odstavci jsou případy užití, které má na starosti sám systém a v čase je autonomně provádí.

- **Automatické vyhodnocení** – Systém provádí rutinu, která zahrnuje zmražení řešení, jenž jeví známky dlouhodobé nečinnosti. Dále sleduje deadline zadání prací a v případě vypršení limitu zadání uzavře a zmrazí práce všech přiřazených řešitelů. Nejdůležitějším úkolem automatického vyhodnocení je, že porovnává odevzdaná řešení pro zadanou práci, přiřazuje jim hodnocení a navrhuje odměnu dle kvality odvedené práce.

3.2 Systém práce

V aplikaci bude fungovat systém správy prací, který zahrnuje zadávání, přiřazování a vyhodnocování.

Vykonání práce může mít podobu kontroly, kdy uživatel pouze kontroluje správnost přepisu a nebo přímo provádí opravy. Zpracovávat se budou buď přepisy slajdů nebo přepisy audia.

Zadání práce obsahuje relevantní informace pro zpracování, referenční data a může se vyskytovat ve třech stavech. U zadání je potřeba rozhodnout, zda může mít dobrovolné řešitele, zda bude volně dostupné, tj. registrovaní uživatelé si jej budou moci vybrat, zda je limitována na počet placených řešitelů a jestli je to práce s odměnou.

- **Nepublikovaná** – zadání práce se může fyzicky nacházet v systému, ale potencionální řešitelé jej neuvidí.
- **Publikovaná** – je viditelná pro potencionální řešitele, dokud nevyprší deadline nebo ji zadavatel manuálně neuzavře.
- **Uzavřená** – nepřijímá nové řešitele, všechny související práce řešitelů jsou již vyřešeny.

Přiřazená práce přejímá kopii referenčních dat ze zadání, musí být jasné, zda se jedná o dobrovolného či placeného řešitele a je-li řešitel anonymní.

- **Zpracovávaná** – V tomto stavu je ihned po přiřazení a setrvává v něm až do řádného dokončení práce.
- **Odevzdaná** – řešitel práci regulérně odevzdal.
- **Zmražená** – nastal nějaký problém, administrátor nebo systém práci zmrazil.
- **Vyhodnocená** – práce byla ohodnocena a pokud se jedná o práci s odměnou, byla za ni řešiteli dle ohodnocení navržena adekvátní odměna.
- **Uzavřená** – pokud byla za práci odměna, administrátor ji odsouhlasil a volitelně provedl záznam o vykonání.

Vyhodnocení prací funguje na principu ohodnocení všech položek zadané práce a odtud se průměrem získá výsledné hodnocení. K hodnocení bude použita následující stupnice (vynikající, velmi dobře, dobře, ucházející, slabé). Práce, které nebyly regulérně odevzdány, ztrácejí nárok na odměnu a nejsou tudíž hodnoceny. Z hlediska výkonnosti získávají uživatelé osobní ohodnocení (nováček, pokročilý, profesionál, expert, guru).

Import a export

- Import/export bude probíhat manuálně.
- Formátem pro přenos dat bude XML.
- Možnost hromadného zpracování.
- Možnost definovat společné atributy individuálním pracím.

Vzhledem k požadavkům na způsob distribuování práce se ve fázi importu i exportu musí zajistit synchronizace mezi daty portálu SuperLectures.com a Anotačního portálu.

Fyzicky se data budou nacházet na SuperLectures.com.

3.3 Konceptuální schéma databáze

Schéma databáze je součástí analýzy, návrhu i implementace datové vrstvy, kompletní schéma viz příloha B.1. Při použití relační databáze jsou entity diagramu mapovány přímo na tabulky. Dále uvádím popis těchto tabulek a vztahů mezi nimi.

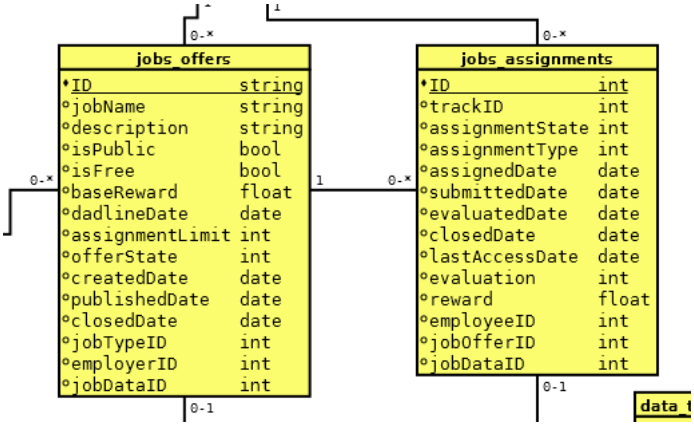
users je tabulka představující konkrétního uživatele, má právě jeden záznam v tabulkách **users_statistics** a **users_settings**, kde je uchována jeho statistika a individuální nastavení. Tabulky by šlo samozřejmě sloučit do jedné, protože u vztahu 1:1 nemá moc smysl vytvářet tabulek více, ale přišlo mi čistější tyto věci od sebe oddělit. Každý uživatel může mít v systému zadané práce, ty jsou ukládány do tabulky **jobs_offers** a jemu přiřazené práce v tabulce **jobs_assignments**. O dokončených pracích je veden záznam v tabulce **jobs_records**. Pokud se jedná o jednorázového uživatele, ten má navíc jednoho administrátora přes práce, který na něj dohlíží.

jobs_offers představuje zadání práce, která je nějakého typu viz tabulka **jobs_types** a má jistý počet řešitelů viz tabulka **jobs_assignments**. Každé zadání práce má uloženo referenční data v tabulce **jobs_data**.

jobs_assignments představuje přiřazení řešitele na konkrétní zadání práce, podobně jako u tabulky **jobs_offers**, data jsou uložena v tabulce **jobs_data**. Tady bych zdůraznil, že se nejedná o jedny a ty samá data, s novým přiřazením práce vzniká i kopie referenčních dat.

Protože jsou předchozí dvě tabulky jádrem aplikace, uvádím na obrázku 3.1 jejich výřez ze schématu.

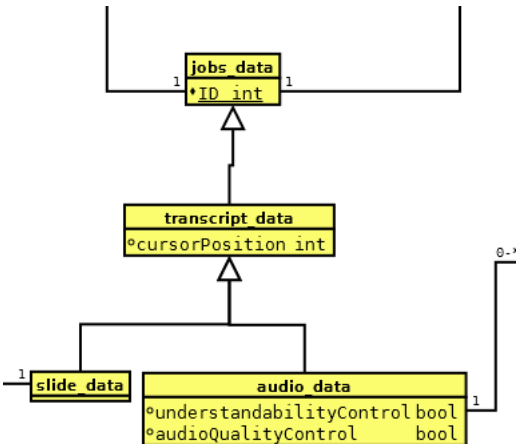
jobs_data - tato tabulka si zaslouží trochu více pozornosti, snažil jsem se tabulku navrhnout tak, aby fungovala jako univerzální úložiště dat pro práce různého typu. Využil jsem přitom specializace a data konkrétního typu práce jsou pak ve vztahu s listovými tabulkami. Zatímco zadání práce a přiřazení práce odkazuje na tabulku generalizovanou viz obrázek 3.2.



Obrázek 3.1: Tabulka *jobs_offers* slouží k uložení dat zadání práce a je ve vztahu 1:n s tabulkou *jobs_assignments*, ta uchovává konkrétní přiřazení práce.

slide_data uchovává nastavení a data práce typu kontrola, oprava přepisů slajdů. Položkami jsou slajdy uložené v tabulce **data_transcript_slides**.

audio_data uchovává nastavení a data práce typu kontrola, oprava přepisů audia. Zde jsou položky jednotlivé segmenty audia, ty jsou v tabulce **data_transcript_segments** a každý segment se skládá z několika slov, pro ty je tabulka **data_transcript_segment_words**. Ukládat každé slovo zvlášť je sice více náročné na operace databázové vrstvy, ale u každého slova je nutné ukládat od kdy, do kdy se v segmentu nachází.



Obrázek 3.2: Generalizovaná tabulka *jobs_data* je společným předkem pro data různých typů prací. Její specializací je tabulka *transcript_data*, ta je zase předkem pro data prací, která jsou vícepoložková a „vícekroková“. Její specializací jsou tabulky *slide_data* a *audio_data*, ty slouží jako obálka nad jednotlivými položkami a uchovávají nastavení pro daný typ práce.

Kapitola 4

Návrh aplikace

V této kapitole se zabývám návrhem jednotlivých vrstev architektury aplikace. Snažil jsem se postupovat v duchu *doménově řízeného návrhu* viz podsekce 2.3.1, ve výsledku byl návrh rychlejší a přehlednější.

Aplikace byla vytvářena iterativně, takže tento návrh lze považovat za poslední cyklus iterace, který byl v rámci této práce dokončen.

Z neformální analýzy lze vytknout tyto moduly, které je potřeba navrhnout a začlenit:

- **Správa uživatelů.**
- **Správa prací.**
- **Vykonávání prací.**
- **Vyhodnocování prací.**

4.1 Architektura aplikace

Aplikace je postavena na klasické třívrstvé architektuře viz obrázek 4.1, která se skládá z vrstvy prezentační, aplikační a datové. A architektonickém vzoru *MVP (Model-View-Presenter)* viz podsekce 2.3.2.

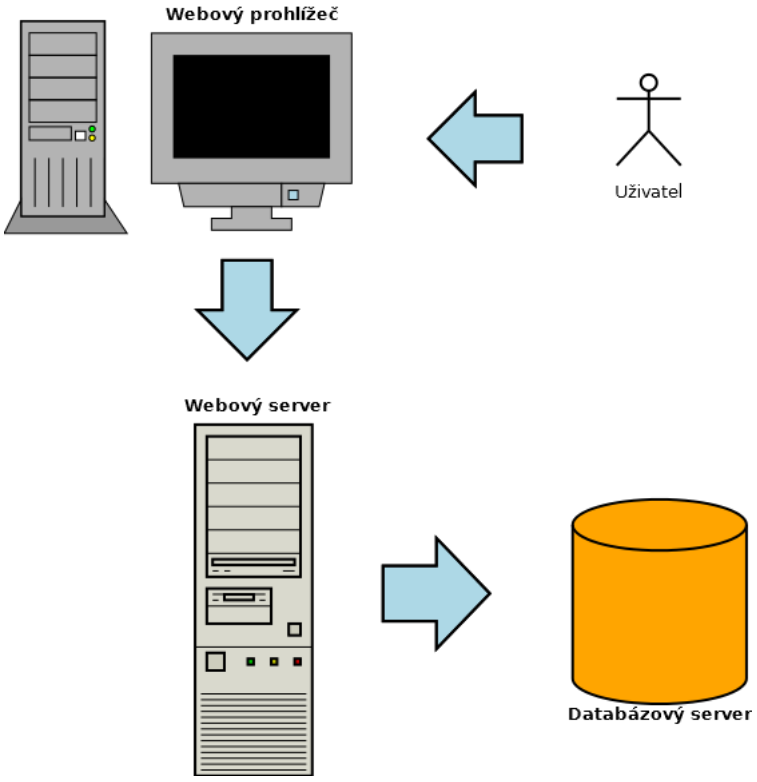
4.1.1 Datová vrstva

Datové vrstvě odpovídá ve vzoru MVP model. Veškeré změny probíhající na této vrstvě jsou výhradně záležitostí modelu.

Součástí návrhu této vrstvy je především návrh modelu viz sekce 4.2. A protože staví na vzoru Entity-Repository-Mapper viz podsekce 2.3.3 znamená to návrh entit a návrh servisní vrstvy. O zbylé vrstvy jako repository, mapper a ovladač datového úložiště se postará ORM framework.

Servisní vrstva má na starosti import/export dat a automatické vyhodnocování. Třídy realizující tyto operace nazývám dále v textu jako služby.

Všechny služby dědí od `BaseService`, která jim poskytne společné závislosti jako je přístup k databázi a případně další. Injektování závislostí je založeno na principu návrhového vzoru *DI (Dependency Injection)*.



Obrázek 4.1: Schéma představující klasickou třívrstvou architekturu.

4.1.2 Aplikační vrstva

Aplikační vrstvě odpovídá ve vzoru MVP model a presenter. Presenter má na starosti tok událostí v aplikaci, zpracování požadavků. Model na této vrstvě přejímá data nebo požadavky od presenteru a podle toho buď aktualizuje svůj stav nebo odpoví na požadavek.

Součástí návrhu této vrstvy je návrh jednotlivých presenterů a jejich akcí viz sekce 4.3. Ty jsou rozděleny do tří modulů, první pro uživatelskou část, druhý pro administrátorskou část a třetí pro vykonání a vyhodnocení práce, tj. vizualizaci dat patřičným způsobem.

4.1.3 Prezentační vrstva

Prezentační vrstvu tvoří ve vzoru MVP view, které má na starosti jak vstup, tak i výstup. Návrh této vrstvy odpovídá návrhu uživatelského rozhraní viz sekce 4.4. Rozhraní lze rozdělit na dvě skupiny, v první skupině jsou ty, které umožní uživateli práci vykonat. Ve druhé ty, které poskytnout administrátorovi náhled na vykonanou práci a podklady pro objektivní vyhodnocení.

4.2 Návrh modelu

Při návrhu modelu jsem použil objektově-relační mapování a konkrétně návrhový vzor *Entity-Repository-Mapper* viz podsekce 2.3.3. K tomu bylo potřeba namapovat tabulky z konceptuálního schématu viz sekce 3.3 na entity ve smyslu ORM a protože model sahá částečně i do vrstvy aplikační, tak definovat uvnitř těchto entit metody reprezentujících

doménovou logiku viz podsekce 4.2.1. Aby byl návrh podle tohoto vzoru kompletní, bylo by potřeba ještě navrhnout mapper a repository vrstvy pro všechny entity, ale to je proprietární návrh tohoto vzoru. Dnes existuje spousta nástrojů, které tyto vrstvy automaticky vygenerují. Pak je tu ještě servisní vrstva, jejíž přítomnost sice není nezbytně nutná, ale zapouzdřuje související operace, takže její použití zpřehledňuje návrh a následnou implementaci. V případě této aplikace patří do servisní vrstvy import/export dat a automatické vyhodnocení.

4.2.1 Návrh entit

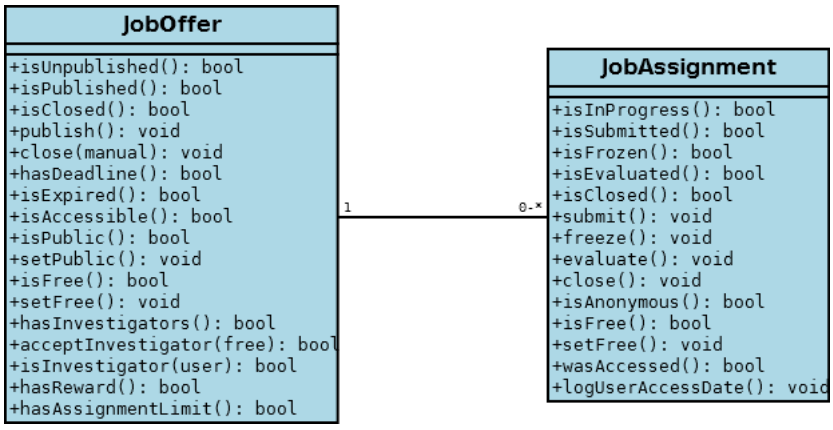
Při vytváření entit objektově relačního mapování stačilo vzít konceptuální schéma a tabulky namapovat na třídy, ve skutečnosti proces probíhá opačně, ale to je záležitostí implementace[1]. Těmto třídám (entitám) stačí definovat metody, které korespondují s pravidly aplikační domény. Např. budu-li chtít změnit stav nějaké entity z odevzdáno na vyhodnoceno, pouze zavolám její metodu.

Kromě toho, že každá entita disponuje tzv.gettery a settery[2], pro přímý přístup k jejím vlastnostem (může jít o atributy tabulky), jsou zde i doménově specifické metody.

Entity mají tu výhodu, že mohou být ve vztahu kompozice, takže budu-li chtít získat všechna řešení k danému zadání, prostě zavolám metodu entity reprezentující zadání práce.

Dále uvádím diagram tříd (entit) pro zadání a přidělení práce. A také způsob manipulace s daty práce. Ostatní entity jsou navrženy podobně, každá nese zodpovědnost za svou část aplikační domény. V diagramech jsem vynechal záměrně atributy, protože jsou shodné s atributy v ER diagramu.

Zadání práce a konkrétní přiřazení viz diagram 4.2.



Obrázek 4.2: Entita *JobOffer* má metody pro správu zadání práce a také pole entit *JobAssignment*, které spravují konkrétní přiřazení práce. U každého zadání jsou tak hned přístupná všechna řešení pouze zavoláním jedné metody.

U entity `JobOffer` jsou metody na dotazování stavu, ve kterém se zadání práce nachází

[1] Podle anotací entit může být vygenerována výsledná struktura databáze.

[2] Gettery a settery jsou metody, pomocí kterých se dá přímo přistupovat k vlastnostem entity, jiný způsob přístupu není správný. Pokud entita nedefinuje setter, lze danou vlastnost pouze číst. Je dobrým zvykem tuto konvenci dodržovat.

jako `isUnpublished(), isPublished(), isClosed()`), Poté metody na změnu stavu – `publish()` zviditelní zadání pro přiřazené řešitele a potencionální řešitele, `close()` uzavře zadání práce a zavolá automatické vyhodnocení přiřazených řešitelů.

Další metody jsou samovypovídající, ale zdůraznil bych jednu typickou metodu DDD návrhu `acceptInvestigator(free = false)`. Ta má za úkol rozhodnout, zda-li je možné k zadání práce přiřadit dalšího řešitele. Musí brát v potaz, jde-li o žádost přiřazení dobrovolného nebo placeného řešitele.

Podobné je to s metodami u `JobAssignment`, pro dotaz na stav `isInProgress(), isSubmitted(), isFrozen(), isEvaluated(), isClosed()`, dále `isAnonymous()` pro zjištění, jde-li o anonymního řešitele (nemá účet v systému). A další...

Manipulace s daty práce viz diagram 4.3.

Entity `SlideData` a `AudioData` slouží jako obálka nad daným typem dat, obsahují specifické nastavení pro zpracování či vyhodnocení. Obě dědí od `TranscriptData`, tato třída je předkem všech takových dat práce, jež obsahují více elementárních položek, jsou tedy vícekrokové. Princip vychází z ER diagramu.

Elementárními položkami jsou pak `Slide, AudioSegment a AudioSegmentWord`.

4.2.2 Import a export dat

U služby pro import/export dat jde především o zpracování XML souborů. V případě importu je to analýza souboru a vytvoření patřičných entit, v případě exportu se jedná o sestavení souboru.

Protože datové položky se liší, je i zpracování souborů odlišné, ale zapouzdření v podobě jedné práce je společné.

Nicméně abstraktní popis funkčnosti je společný:

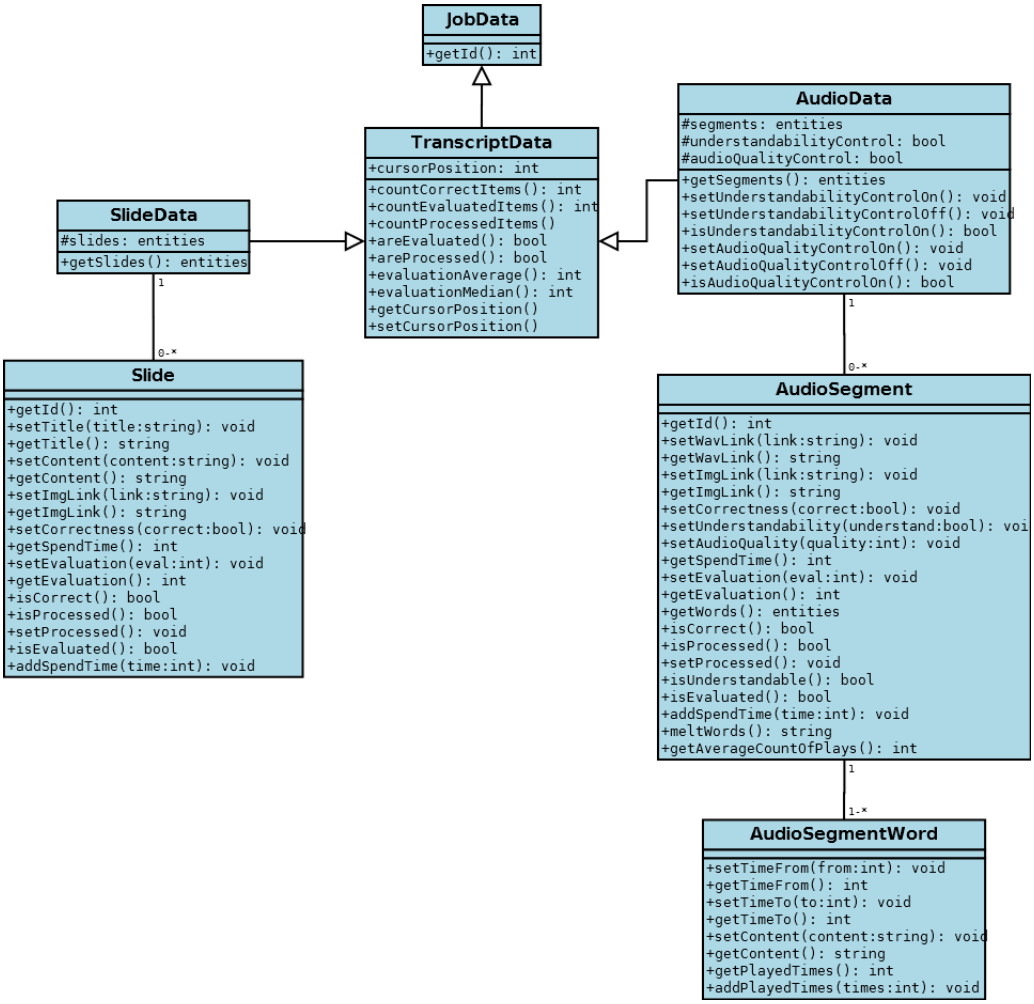
Import je rozdělen na dvě fáze, analýza a vlastní zpracování. Ve fázi analýzy probíhá kontrola struktury souboru s daty, tj. přítomnost povinných položek, validní XML formát a když v této fázi nastane chyba, import jednodušše skončí a oznámí chybu uživateli. Druhou fází je import vlastních dat, tzn. vytvoření entit, vzájemných vazeb a uložení do databáze.

Obě fáze využívají kaskádové definice položek. Takže v souboru je společná hlavička s položkami pro všechny práce, ale individuální práce může tyto položky „přetížit“ vlastními. Kaskáda jde ještě o úroveň výše a to na úroveň zadání položek do formuláře uživatelem. Přetížení jde ve směru od položek z formuláře, přes společnou hlavičku v souboru až po individuální hlavičku zadání práce.

Export je o něco jednodušší v tom smyslu, že není potřeba více fází. Spočívá pouze v transportu dat směrem z databáze do XML souboru.

Tento návrh funkčnosti zajišťuje možnost importovat či exportovat jednu práci i vícero zároveň.

V příloze můžete shlédnout formát přenášených dat pro práce typu přepis slajdů viz příloha C.1 a přepis audia viz příloha C.2. Jde pouze o ukázku struktury, nikoliv konkrétní příklad.



Obrázek 4.3: Návrh entit pro práci s uloženými daty je odrazem konceptuálního schématu databáze. Každá entita na schématu disponuje metodami pro tu část hierarchie, kterou v rámci tohoto uspořádání zastává.

4.2.3 Automatické vyhodnocení

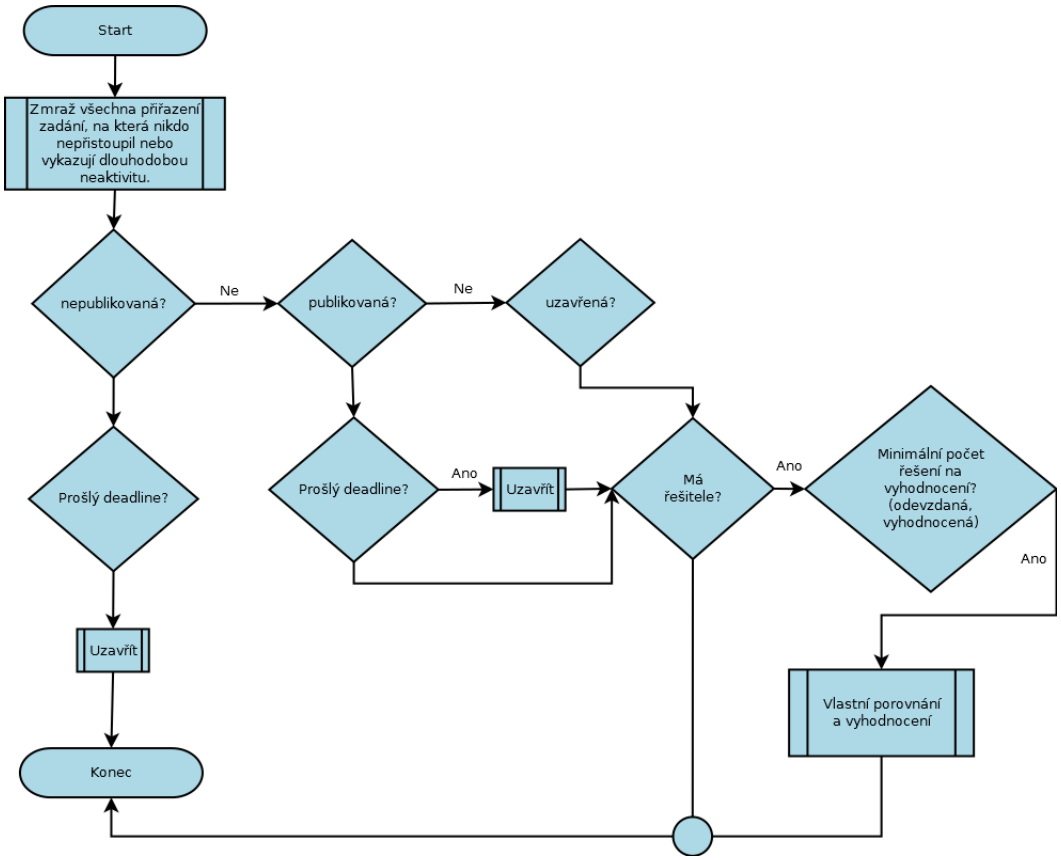
Aby administrátor nemusel všechno hlídat ručně, navrhl jsem službu pro automatické vyhodnocení. Služba může být spuštěna manuálně nad konkrétní prací a nebo jako rutina pomocí systémového démona jako je třeba CRON [3]. Má za úkol hlídat deadline u zadaných prací, případně je uzavírat. Dále průběžně hodnotit příslušná řešení, tj. odfiltrovat ta, která jsou stále rozpracovaná, ale dlouhodobě nejeví známky aktivity a dále zadávat hodnocení těm, které čekají na vyhodnocení.

Základní myšlenkou je vzájemné porovnávání regulérně odevzdaných řešení čekajících na hodnocení a již hodnocených řešení. Hodnocená řešení již nejsou znovu přehodnocena, ale slouží k objektivnějšímu porovnání, čím více řešení, tím přesnější hodnocení lze získat.

Služba má tři fáze, od obecné týkající se jakékoliv práce, až po specifické vyhodnocení. Uvedu několik schémat znázorňujících princip jejich funkčnosti.

[3] Jako rutina prochází všechny práce v systému a spouští vyhodnocení nad každou z nich.

1.fází je kontrola stavu, deadlinů a případné uzavření zadání, aby již nemohlo být dále přiřazováno a s tím souvisí zmražení všech řešení, která byla v době uzavření rozpracovaná. Zároveň je provedeno zmražení všech řešení, která jsou stále rozpracovaná a nejeví dlouhodobě aktivitu. Následuje rozhodnutí dle stavu o přechodu do další fáze vyhodnocení. Tuto fázi znázorňuje diagram viz obrázek 4.4.



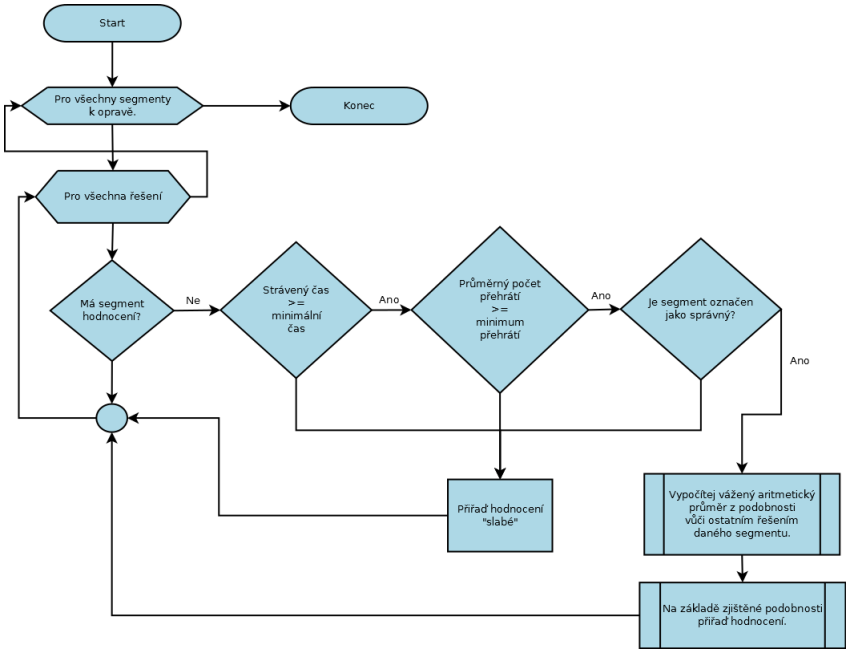
Obrázek 4.4: Vývojový diagram pro 1.fázi automatického vyhodnocení.

2.fází je specifické vyhodnocení konkrétního typu práce. Pro ukázku jsem zvolil opravu přepisů audio segmentů. Na diagramu 4.5 můžete vidět průběh této fáze.

Vyzdvihnu-li hlavní myšlenku tak, dochází k porovnání všech řešení pro každý segment zvlášť. Může se stát, že již hodnocení má, pak se přeskočí, jinak se pokračuje na odfiltrování nekvalitně provedených prací. Tj. uživatel strávil na segmentu podezřele krátký čas, nepřehrál si vůbec segment a nebo neoznačil segment jako opravený. Takový segment získává nejhorší hodnocení. Projde-li segment tímto filtrováním, dochází k vlastnímu porovnání vůči ostatním.

Díky filtrování se může stát, že přestane platit podmínka minimálního počtu řešení pro automatické vyhodnocení. V tomto případě není hodnocení segmentu uděleno.

Jinak je vypočítána podobnost daného segmentu vůči ostatním (textové opravy přepisu), na procentuální podobnost dvou řetězců lze použít klasické algoritmy z knihovny daného jazyka. V mém případě je to funkce `similar_text()` z knihovny jazyka PHP. Ze získaných podobností je vypočítán vážený aritmetický průměr viz vzorec 4.1. Váhy tvoří



Obrázek 4.5: Vývojový diagram pro 2.fázi automatického vyhodnocení.

osobní ohodnocení řešitele, tj. ti s vyšším ohodnocením mají podstatně vyšší váhu a také stav, případně hodnocení daného segmentu.

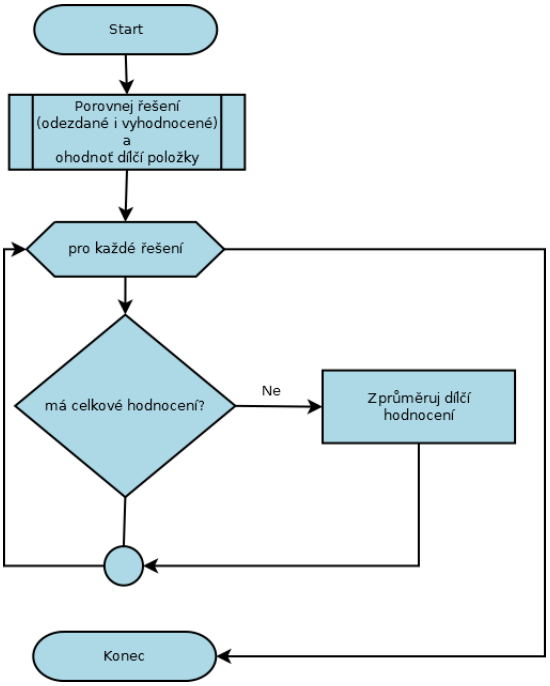
$$\overline{x} = \frac{\sum_{i=1}^{n} w_i * x_i}{\sum_{i=1}^{n} w_i} \tag{4.1}$$

Výsledné hodnocení se odvíjí od získaného váženého průměru ze všech podobností pro daný segment, procentuální rozsah je namapován na škálu hodnocení. Na závěr této fáze bych podotknul, že je velice citlivá na nastavení jednotlivých váh a závěrečného mapování procentuálního rozsahu na škálu hodnocení.

3.fází je zprůměrování dílčích hodnocení každého řešení viz vzorec 4.2 a stanovení výsledného hodnocení.

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{4.2}$$

Retrospektivně tato fáze využívá fázi druhou pro získání právě těch dílčích hodnocení, kde se mohlo stát, že byla porušena podmínka minimálního počtu řešení pro automatické vyhodnocení. A to vede k tomu, že nebyla stanovena všechna dílčí hodnocení a práce nemůže nabýt výsledného hodnocení. Každopádně jednou stanovená dílčí hodnocení zůstávají, zbytek může být doplněn při dalším pokusu. Činnost můžete vidět na diagramu viz obrázek 4.6.



Obrázek 4.6: Vývojový diagram pro 3.fázi automatického vyhodnocení.

4.3 Návrh presenterů

Základem je rozdělení celé aplikace do modulů, rozdělení na uživatelskou a administrátorskou část je samozřejmostí, předchází se tím neustálým kontrolám oprávnění uživatelů na prováděné akce. Pak je tu ještě společný modul pro vizualizaci práce využívaný jak pro zpracování, tak vyhodnocování práce.

Uživatelský modul

- UserPresenter – akce spojené s uživatelským účtem, změna hesla, individuální nastavení.
- JobPresenter – akce, které je potřeba provést před přesměrováním na vykonávání práce.
- SignPresenter – stará se o registraci, přihlašování a odhlašování uživatelů ze systému.

Administrátorský modul

- DashboardPresenter – denní, dlouhodobá statistika aktivit v systému, rychlý přístup k častým operacím.
- UserPresenter – akce jako přidání nového uživatele, editace stávajícího uživatele, smazání uživatele a zobrazení informací o uživateli.
- JobPresenter – akce spojené se zadáním práce jako import, smazání, uzavření, přiřazení řešitele, dále akce spojené s konkrétním řešením jako prohlídka, smazání, předání jiném řešiteli na přepracování, vyhodnocení a export.

Modul pro vizualizaci práce

- SlideTranscriptPresenter – vizualizace práce typu kontrola, oprava slajdů pro zpracování, vyhodnocení.
- AudioTranscriptPresenter – vizualizace práce typu kontrola, oprava audia pro zpracování, vyhodnocení.

4.4 Návrh uživatelského rozhraní

Při návrhu uživatelského rozhraní jsem se zaměřil na rozhraní pro vykonání a vyhodnocení práce, to je alfou a omegou tohoto projektu. Aplikace musí poskytnout příjemné a jednoduché prostředí motivující uživatele strávit nad prací nějaký čas.

V systému se nacházejí dva typy dat ke zpracování a k tomu dva ač podobné, ale odlišné způsoby zpracování, kontrola nebo oprava. Přišlo mi vhodné využít pro kontrolu i opravu stejného rozhraní, ale s drobnými změnami funkčnosti. Prvků tohoto rozhraní využívá i rozhraní pro vyhodnocení, je pouze okořeněno o další funkcionalitu převážně za účelem objektivního hodnocení.

Dále uvádím návrh těchto rozhraní.

4.4.1 Vykonávání práce

U obou typů prací se uživatel bude pohybovat po jednotlivých položkách a provádět buď kontrolu nebo opravy. Bude využívat následujících ovládacích prvků.

- Vizualizace položek ke zpracování v podobě *miniaturní grafické reprezentace seznamem*, takže uživatel přímo uvidí, kde se nachází a jestli již danou položku zpracoval. V seznamu bude graficky reprezentována podle stavů **nezpracována/správně/špatně**. Kliknutím na konkrétní položku v seznamu se dostane na žádanou pozici.
- Na další položku se může dostat pomocí tlačítek *správně/špatně*, čímž udává nový stav předchozí položky.
- Může využít klávesových zkratek, pro někoho je tato varianta pohodlnější než neustálé klikání myší.

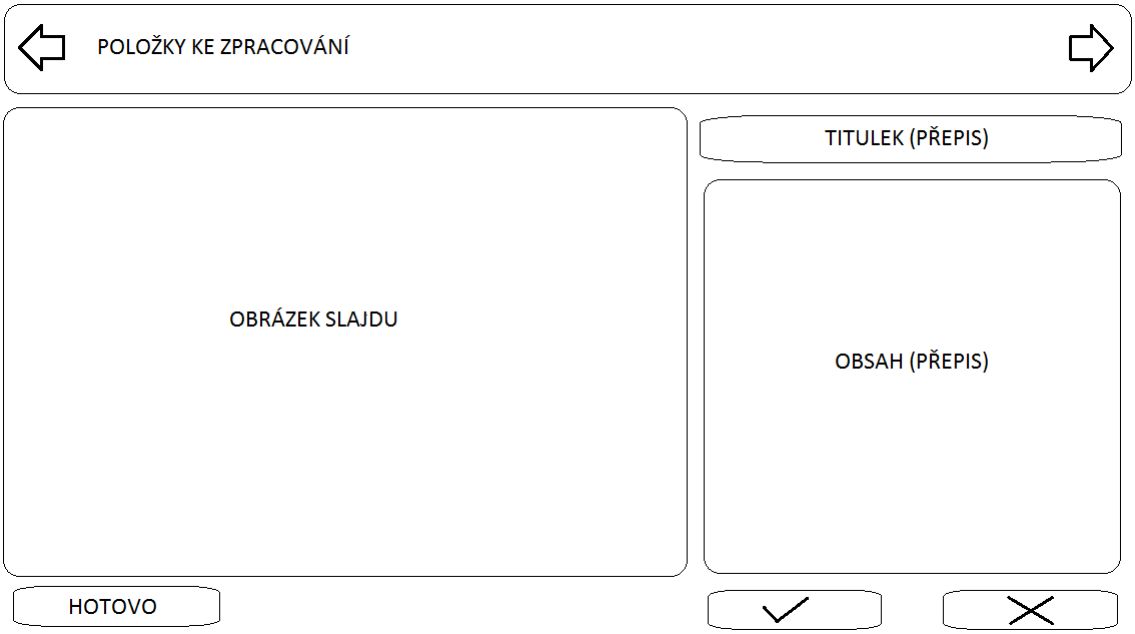
Půjde-li o kontrolu správnosti přepisů, nemá uživatel možnost přepis nijak měnit.

Kontrola, oprava přepisů slajdů U rozhraní pro kontrolu, opravu přepisu slajdů stačí dát uživateli k dispozici obrázek zpracovávaného slajdu a jeho přepis, ten bude rozdělen na titulek slajdu a vlastní obsah.

Na obrázku 4.7 lze vidět odpovídající rozložení prvků.

Kontrola, oprava přepisů audia Rozhraní pro kontrolu, opravu přepisů audia je více komplikovanější. Uživatel bude mít k dispozici obrázek představující zvukovou stopu audia, vlastní přepis jednotlivých slov a ovládací prvky pro přehrávání a volbu šíře kontextu v jednotkách slov.

- Pro přehrátí může využít klasických ovládacích prvků pro přehrávač.



Obrázek 4.7: Návrh uživatelského rozhraní pro zpracování přepisů slajdů.

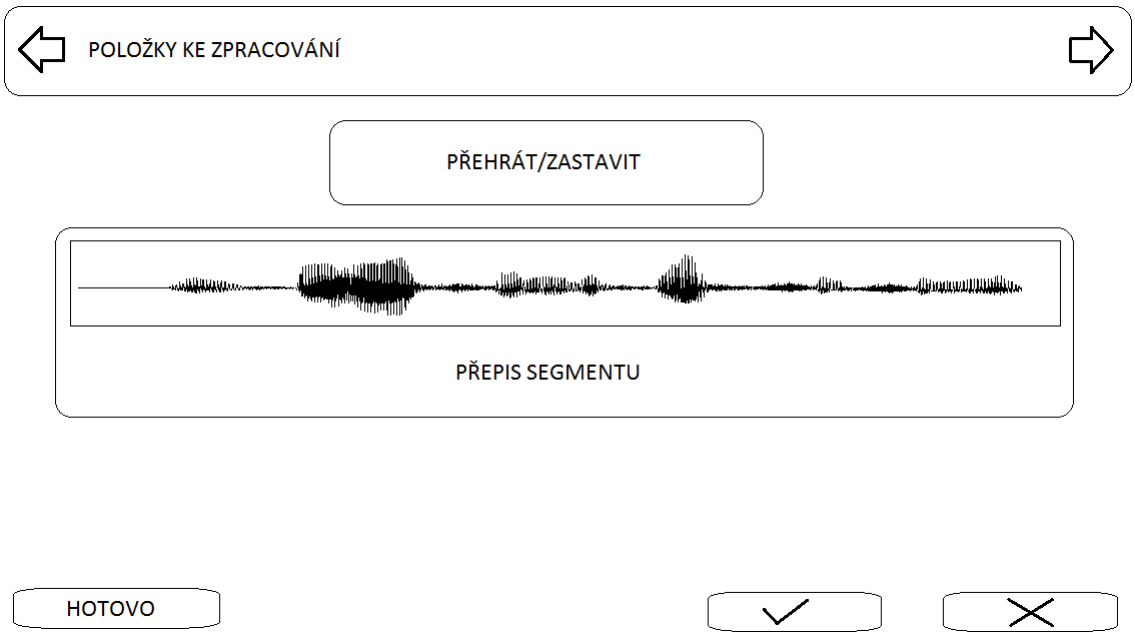
- Může si také přehrát konkrétní slovo plus zvolený kontext kliknutím do obrázku nebo využitím klávesových zkratek.

Na obrázku 4.8 lze vidět odpovídající rozložení prvků.

4.4.2 Vyhodnocení práce

Protože návrh vychází z předešlého, zmíním pouze vzájemné odlišnosti a přidanou funkcionalitu.

- V seznamu miniatur jednotlivých položek přibyl další možný stav a to „ohodnoceno“. Administrátor tak má přehled o položkách, které již mají hodnocení.
- Přibyla škála hodnocení.
- U vyhodnocení práce typu kontrola, oprava přepisů slajdů má administrátor k dispozici pohled na změny oproti původnímu zadání. K tomu má k dispozici informaci o čase, který řešitel nad daným slajdem strávil.
- U vyhodnocení práce typu kontrola, oprava přepisů audia má navíc k dispozici graf reprezentující počet přehrátí každého slova v segmentu.



Obrázek 4.8: Návrh uživatelského rozhraní pro zpracování přepisů audia.

Kapitola 5

Implementace a testování

V této kapitole se zabývám implementací struktury, komponent a funkčnosti aplikace zmíněných v předešlém návrhu. Zaměřuji se spíše na ty, které si zaslouží podrobnější vysvětlení. Za zmínku určitě stojí některá uskalí, na která jsem v průběhu implementace narazil a jejich řešení. Dále popisuji, co bylo potřeba udělat před nasazením a na závěr testování a jeho výsledky.

Implementaci jsem provedl za použití nástrojů uvedených v 2.4. S jazykem PHP jsem se seznámil v knize [1] a konkrétně s jeho objektově orientovaným využitím v [11].

5.1 Struktura aplikace

Strukturu aplikace definuje už samotný vzor *Model-View-Presenter* a *Nette*, na kterém je postavena. Začnu tedy od hieararchie presenterů, popisu šablonovacího systému až po začlenění modelu do struktury.

Protože cílem této kapitoly není vysvětlit podrobně, jak Nette implementuje vzor MVP, tak bych vás rád odkázal na oficiální dokumentaci, kde je vysvětlen životní cyklus presenteru `http://doc.nette.org/cs/presenters#`. Nepovažuji tuto znalost za zcela nezbytnou pro pochopení dalšího textu, ale přispívá rozhodně k lepší orientaci.

5.1.1 Hierarchie presenterů

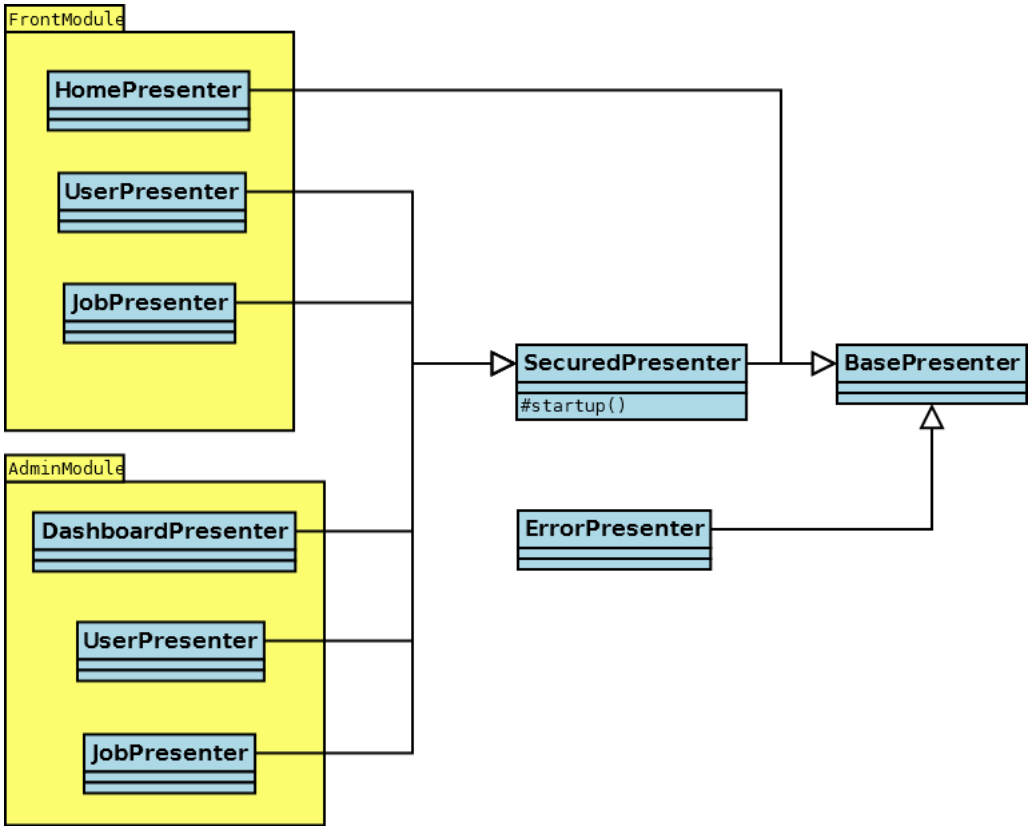
Předkem všech presenterů je `BasePresenter`, ten dědí od `Nette\Application\UI\Presenter`. A jsou v něm definovány operace společné pro všechny presentery. Dalším důležitým presenterem je `SecuredPresenter`, ten poskytuje autentizaci a autorizaci pro všechny další presentery, kde je nežádoucí nekontrolovaný přístup[1]. Nezávislým presenterem je `ErrorPresenter`, ten je součástí Nette a je vyvolán, pokud nastane nějaký problém.

Na obrázku 5.1 jsou vidět i zbývající presentery, jejich funkcionalita vychází z návrhu.

5.1.2 Šablony akcí

Nette disponuje schopným systémem šablonování nazvaným Latte, jehož předností je využití maker a tzv. helperů. Největší výhodou je ale eliminace duplikace kódu, každá aplikace má nějaké části svých šablon společné a Latte tyto společné části řeší pomocí tzv. „layoutů“.

[1] Presenter neobsahuje přímo logiku autentizace a autorizace, ale volá služby, které ji implementují a to `Models\Security\Authenticator` a `Models\Security\Authentizator`



Obrázek 5.1: Schéma zobrazující implementovanou hierarchii presenterů.

V aplikaci využívám 1 společný layout pro všechny moduly, ten definuje kostru. A její obsah tvoří dílčí šablony, které se do ní vkládají. Každá taková šablona odpovídá nějaké akci presenteru, ovšem může se stát, že má více akcí společnou právě jednu šablonu.

5.1.3 Integrace modelu

Model zde představuje využívání služeb definovaných výše v návrhu a přímou interakci s entitami. Aby se dalo ke všemu lehce přistupovat, využil jsem třídy `Nette\DI\Container` ta má tu vlastnost, že do ní lze vložit všechny služby, které jsou používány napříč aplikací. Přes tento kontejner tedy přistupuji přímo k Doctrine2 entity manažeru a již zmiňovaným službám. Odpadla tak potřeba na každém místě znovu vytvářet instance těchto tříd.

5.2 Implementace volného přístupu

Za volný přístup k zadání práce označuji pokus o vypracování bez explicitního přidělení a neziskově. Většinou se jedná o anonymní návštěvníky z portálu SuperLectures.com, ale může jít i o registrované uživatele, kteří se rozhodnou vypomoci.

Pokud se navíc anonym zaregistruje, je žádoucí, aby byly všechny jeho doposud vykonané práce navázány na jeho nový účet, právě proto je potřeba ho nějak identifikovat.

Je tedy potřeba sledovat, které práce konkrétní anonym vykonal. U registrovaného uživatele to není problém, práce je mu implicitně přidělena a přímo navázána na jeho účet.

Toto sledování jsem implementoval pomocí cookies. Entita `JobAssignment` spravuje identifikátor pro sledování, který není totožný s identifikátorem entity. Díky tomu lze rozhodnout, zda jde o anonymního řešitele. Stejný identifikátor je zasílán anonymem v cookie, pokud si tedy nesmaže všechny cookies uložené v prohlížeči, dají se spolehlivě dohledat jeho řešení.

Přístup k zadání práce ze SuperLectures.com má následující tvar `http://anotace.prepisovatel.cz/job/free-processing/identifikator`. Jedná se o identitikátor entity `JobOffer`, podle přihlášení uživatele nebo zaslaného identifikátoru v cookie lze dohledat jeho konkrétní přiřazení (řešení). Při prvním přístupu na takové zadání je uživateli práce implicitně přiřazena, to zahrnuje kopii referenčních dat a vytvoření záznamu v podobě `JobAssignment`.

Bezpečnost zde hraje podstatnou roli, protože přístup k pracím přes URL je přes identifikátory práce, musel jsem je implementovat jako hash, to znesnadní přístup hravým uživatelům. Stejně tak je hash identifikace u entity `JobAssignment`, na jedno konkrétní přiřazení nelze tedy jednodušše přistoupit[2].

5.3 Implementace služeb

Jako předka všech služeb jsem položil třídu `BaseService`, ta jim poskytuje všechny potřebné závislost, hlavně přístup k databázi.

Import/export mají na starosti služby `SlideDataService` a `AudioDataService`. Obě implementují princip viz podsekce 4.2.2. Při jejich implementaci jsem narazil hned na několik problémů. Protože těmito službami prochází kvanta dat, tak se zvyšují i nároky na dobu zpracování a systémovou paměť. V prvních iteracích jsem využíval pro import pouze knihovny *PHP DOM*, která načte celý soubor do paměti a pak ho lze pohodlně procházet. Navíc jsem neprováděl odděleně analýzu dat a vlastní zpracování, což vedlo k nepříjemným situacím, kdy se v polovině zpracování objevila chyba a všechna doposud zpracovaná data musela být zahozena. Tento přístup nebyl správný z hlediska doby zpracování a konzumace systémové paměti.

Řešením bylo provést při importu dvě fáze, zvlášť analýzu a zvlášť vlastní zpracování, tak se případná chyba objeví hned a není nutné mazat již zpracovaná data z databáze[3]. Kromě toho nelze načítat celý soubor do paměti pomocí DOM, udělal jsem tedy kompromis a pro sekvenční průchod použil *PHP XMLReader* a pouze zpracování jednotlivých prací jsem nechal na DOM, protože se s ním lépe pracuje.

Paměťové nároky tak podstatně klesly a 128MB paměti pro běh skriptu by mělo stačit i na rozsáhlé importy. Doba zpracování se také zlepšila, nicméně zpracování 10MB souboru si vyžádá i několik minut. To už je ale daň za použití Doctrine2 a objektově-relačního mapování. Obecně ORM není příliš vhodné na hromadné zpracování právě z důvodů paměťové náročnosti. Při každém použití nějaké entity totiž zůstává po dobu běhu skriptu v paměti

[2] U entity `JobAssignment` je využito hashe především kvůli tomu, že uživatelské rozhraní pro vykonání práce díky volnému přístupu nemůže využívat služeb autentizace a autorizace. Hravý uživatel by tak mohl zkoušet číselné kombinace, dostat se na řešení někoho jiného a napáchat škody.

[3] Při tak velkém objemu dat bohužel nelze pokrýt celý import jednou transakcí, transakce tedy obsáhne vždy pouze jednu konkrétní práci.

a tak může dojít k brzkému vyčerpání limitu[4]. Ale i zde existuje řešení v podobě promítnutí změn do databáze po zpracování n položek, tím se paměť uvolní a zpracování dalších položek může pokračovat.

Automatické vyhodnocení obstarávají služby `CommonEvaluationService`, která má na starosti 1.fázi z návrhu viz podsekce 4.2.3. 2. a 3. fázi obstarává `SlideEvaluationService` a `AudioEvaluationService`. Ty mají společného předka `BaseEvaluationService`, kde jsou definovány potřebné váhy, výpočet vah a 2.fáze vyhodnocení.

5.4 Implementace uživatelského rozhraní

Jak jsem již zmiňoval v sekci o struktuře aplikace viz 5.1.2, prostředí aplikace je postaveno na kostře a její obsah je tvořen dílčími šablonami. Protože cílem nebylo zabývat se přímo webdesignem, pro tuto kostru jsem využil administrátorské šablony. Součástí této šablony je pouze základní struktura definovaná pomocí HTML a vytvoření stylů pomocí CSS. Veškerou další funkcionalitu jsem musel implementovat. Ukázku administrátorské obrazovky pro správu prací můžete shlédnout v příloze D.5. Zbývající rozhraní pro vykonání a vyhodnocení práce jsem navrhl a implementoval.

Všechny rozhraní jsem implementoval jako komponenty[5], ty zapouzdřují svou funkcionalitu, takže se nestane, že by se „nafukovala“ logika presenterů.

Původně jsem zamýšlel, že implementuji předka pro komponenty vykonávání a vyhodnocení práce, ale ukázalo se, že bych se tím moc duplicit nezbavil, navíc by přibylo spousta větvení, takže jsem implementoval pro každý účel komponentu zvlášť. Součástí implementace takové komponenty bylo vytvořit její třídu a příslušnou šablonu, která se bude vykreslovat.

Tyto komponenty jsou pak připojeny k presenteru a je jim předána entita `JobAssignment`, která představuje konkrétní přiřazení a tudíž i potřebná data.

V dalším textu popisuji, jak jsou tyto komponenty implementovány, ale nejdříve, co mají společného.

Všechny komponenty obsahují komponentu formuláře, ten slouží pro zobrazení a manipulaci s přepisy. Mají-li být přepisy pouze zobrazeny, pak je editace prvků tohoto formuláře zablokována. Komponenty pro vykonávání práce mají dva módy, prvním je **kontrola** a druhým **oprava** přepisů, u prvního je právě znemožněna editace formuláře, slouží pouze k zobrazení. Stejně tak je tomu u komponenty vyhodnocení práce, první mód slouží jako **inspekce**, kdy si administrátor může prohlídnout doposud nedokončenou práci, ale nemůže zadávat hodnocení a nic měnit. Druhým módem je vlastní **vyhodnocení**, tady má již dovoleno zadávat hodnocení.

Komponenty pro vykonání práce tvoří třídy `ProcessingInterface`[6]. Ty mají metody pro zaznamenání statistik uživatele, zpracování a vyhodnocení dat z formuláře (především pohyb mezi jednotlivými položkami) a pro naplnění šablony aktuálními daty.

[4] Může za to koncepce vzoru *Unit of Work*, na kterém jsou objektově-relační nástroje postaveny viz podsekce 2.3.3

[5] V Nette je vykreslitelnou komponentou označena třída, která dědí od `Nette\Application\UI\Control`, taková komponenta je znovupoužitelná. Součástí jedné komponenty může být další, to je další dobrou vlastností Nette, využívá hierarchie komponent.

[6] Jmenují se stejně, ale jsou odlišeny jmennými prostory.

V šablonách využívají objektu `Logger`, ten slouží k zaznamenání času, který uživatel na dané položce strávil a u přepisů audia navíc k zaznamenání počtu přehrátí jednotlivých slov. U přepisů audia jsem také implementoval v JavaScriptu objekt `Player`, který je založen na HTML5 audio elementu. Ten umožnil vytvoření vlastního ovládání, jak fyzického, tak klávesových zkratek a dále propojení s obrázkem reprezentujícím zvukovou stopu segmentu. Kromě vzájemné provázanosti obrázku se zvukovou stopou a audiem, jsem musel synchronizovat zvýrazňování textu přepisu s aktuálně přehrávaným slovem. Toho jsem dosáhl pomocí javascriptových časovačů. Využil jsem dva, jeden pro zmiňované zvýrazňování textu a druhý pro přehrání pouze konkrétního slova se zvoleným kontextem.

Většina prvků v šablonách komponent využívá knihovny *jQuery UI* a dalších pluginů postavených na *jQuery*. Dále jsou zde implementovány obsluhující funkce pro AJAXové požadavky, klávesové zkratky a podobně. Využití AJAXu mi zde přišlo vhodné, protože není potřeba při pohybu mezi položkami načítat vše, ale pouze data konkrétní položky.

Výsledná rozhraní můžete shlédnout v příloze, rozhraní pro přepis slajdů viz příloha D.1 a rozhraní pro přepis audia viz příloha D.3.

Komponenty pro vyhodnocení práce tvoří třídy `EvaluationInterface`. Nezaznamenává se zde žádná statistika a funkčnost je upravena pro zobrazení relevantních dat k vyhodnocení (statistika, rozdíl přepisu oproti referenčním datům) a zadání vlastního hodnocení (změna formulářových prvků – škála hodnocení).

V šablonách je navíc využito JavaScriptové knihovny pro vizualizaci rozdílu textu, to umožňuje administrátorovi přímý pohled na to, co vlastně uživatel udělal v případě oprav.

Výsledná rozhraní můžete shlédnout v příloze, rozhraní pro přepis slajdů viz příloha D.2 a rozhraní pro přepis audia viz příloha D.4.

5.5 Testování

Protože byla aplikace vytvářena iterativně, průběžné testování udávalo další směr. Zpočátku šlo především o pre-alpha testování viz podsekce 5.5.1, kdy jsem při vytváření aplikace sám testoval jednotlivé komponenty, v pokročilejších verzích byly zapojeni i lidé z výzkumné skupiny BUT Speech@FIT viz podsekce 5.5.2 a v posledních verzích jsem nechal otestovat výslednou aplikaci veřejnost, především návštěvníky portálu SuperLectures.com viz podsekce 5.5.3. Z veřejného testování vyplynula statistika viz podsekce 5.5.4 a heatmapa práce uživatele s uživatelským rozhraním viz podsekce 5.5.5.

5.5.1 Pre-alpha

Tato fáze zahrnovala analýzu, návrh i implementaci aplikace. Jednalo se o prvotní verzi, která posloužila jako základ pro další pokročilejší verze. Původně měla mít aplikace trochu jiné využití viz možná rozšíření v požadavcích v sekci 2.1.1, tomu jsem také přizpůsobil návrh. Díky tomu se v aplikaci nachází některé role jako (administrátor přes práce, jednorázový uživatel a regulární uživatel), které zatím nejsou využité. Každopádně systém správy uživatelů, přidělování práce našel využití i v dalších verzích.

S webovými technologiemi jsem se v tomto rozsahu setkal poprvé, proto mi tato fáze vzala dosti času, musel jsem nastudovat všechny potřebné technologie, postupy, aplikaci navrhnout a uvést poprvé do života.

5.5.2 Alpha

Alfa testování už bylo ve smyslu spolupráce s vývojáři SuperLectures.com. Aplikace postupně získávala podobu, kterou má nyní. Další kroky směřovaly k větší provázanosti s portálem SuperLectures.com, proto bylo potřeba opakovaně upravovat návrh a zapracovat změny. Časté změny návrhu, které zasahovaly i do databázové vrstvy, mě přiměly použít systém migrací, kterým Doctrine2 disponuje. Mám tak přehled o všech dílčích změnách, které byly provedeny s možností návratu.

Dále probíhalo aktivní testování uživatelského rozhraní. Právě feedback z této fáze testování mi poskytl dostatek informací k tomu, aby se po následných úpravách dala aplikace opravdu použít. Před následující fázi beta testování byl implementován sběr statistik chování uživatele, tj. doba strávená na kontrole, opravě a u přepisů audia počet přehrátí jednotlivých slov.

5.5.3 Beta

V poslední veřejné fázi testování došlo k ostrému nasazení na webhosting `http://anotace.prepisovatel.cz/www`. Do aplikace byla vložena data přepisů z říjnových přednášek Barcampu v roce 2011. A na SuperLectures.com byly na stránky s touto konferencí viz `http://www.superlectures.com/barcampbrno2011/` přidány odkazy na opravy přepisů audia s využitím Anotačního portálu viz příloha E.1.

V rámci sledování statistik byl Anotační portál přidán na seznam sledovaných webů v rámci SuperLectures.com pomocí *Google Analytics*. Aby bylo zřetelné, jak uživatel pracuje s uživatelským rozhraním, začlenil jsem do aplikace interní heatmapu, která umožňuje zaznamenat činnost návštěvníků při používání uživatelského rozhraní v rámci každého dne od data nasazení.

Aplikace byla nasazena pro veřejné testování kolem 15.4.2012, od té doby probíhá sběr dostatečného počtu řešení oprav přednášek Barcampu, které budou následně vyhodnoceny a použity na SuperLectures.com.

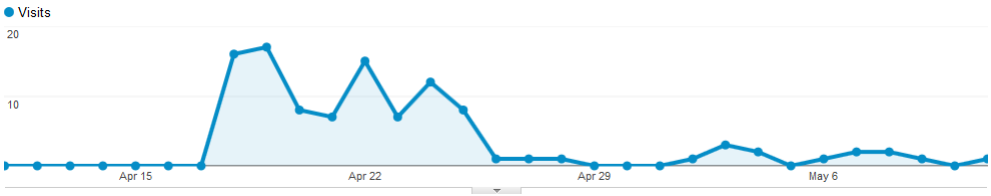
Dále uvádím statistiky náštěvnosti a zpracování dat od spuštění a heatmapu při opravách přepisů audia.

5.5.4 Statistiky

Začnu celkovou návštěvností, portál zaznamenal 106 návštěv viz graf na obrázku 5.2, z toho 30 unikátních uživatelů. Aktivita uživatelů byla na vrcholu od půlky dubna a utichla postupně těsně před koncem dubna. Většina uživatelů pochází z České Republiky, ale našlo se i pár ze Spojených států a Německa. Průměrná doba návštěvy činí 17 minut. Z této doby se dá usoudit, že někteří opravdu pracují. 75 % z nich využívá Firefox, 14 % Chrome, 6 % Internet Explorer a zbytek připadá na Operu a Safari. 45,28 % návštěv je odkazem ze SuperLectures.com a 54.72 % jsou přímé návštěvy (pravděpodobně návštěvy z alfa fáze testování).

V systému se nachází 1116 elementárních prací, 140 rozdělaných řešení a 17 odevzdaných, čekajících na vyhodnocení. Z těchto čísel se dá vyčíst, že spousta uživatelů na práci pouze nahlédne, ale nic neopraví. Tyto řešení jsou následně odfiltrovány při automatickém vyhodnocení.

Důvodem pro nízký zájem potencionálních řešitelů je jistě chybějící finanční či jiná motivace, ale občas se najdou tací, kteří se rozhodnou vypomoci.



Obrázek 5.2: Graf reprezentující návštěvnost od ostrého nasazení.

5.5.5 Heatmap

Z obrázku s heatmapou viz obrázek 5.3 vyplývá, že uživatel se s rozhraním pro vykonávání práce zžil. Nekliká zmateně mimo cílenou oblast a pravděpodobně svému úkolu rozumí. Při opravách využívá kromě běžného přehrátí i přehrátí konkrétního slova, tedy kliká přímo do obrázku. Pro pohyb mezi jednotlivými položkami nevyužívá přímo miniaturního seznamu všech položek, ale označí segment za správný a tím se dostává na další. Protože počet kliků na heatmapě není nijak vysoký, uživatel nejspíše využívá i klávesové zkratky.

Obrázek 5.3: Heatmapa práce uživatele při opravě přepisů audia.

Kapitola 6

Závěr

Cílem této práce bylo vytvořit aplikaci, která umožní provádět ruční opravy automaticky zpracovávaných dat. Těmito daty jsou především textové přepisy slajdů a audia u přednášek na portálu SuperLectures.com. Návštěvník, který si pouští přednášku a narazí na nějakou chybu, má možnost vypomoci a daný úsek opravit. Poté je přesměrován na Anotační portál, jak byla tato aplikace pojmenována, kde je daný úsek přednášky s chybou zadán jako elementární práce. Každý takový úsek může být opraven více uživateli, jejich výstupy jsou potom porovnávány, hodnoceny a jejich sloučením vzniká kvalitní použitelná oprava.

Tento způsob vykonávání práce je postaven na principech crowdsourcingu, který přímo využívá veřejnosti za účelem zisku mnoha řešení, ale s nízkými náklady.

Aby byla aplikace snadno dostupná a umožňila realizovat principy výše zmiňovaného crowdsourcingu, navrhl jsem ji jako webovou aplikaci. Pokud by byla koncipována jako klasická desktopová aplikace, bylo by potřeba provádět individuální instalace a to by mohlo mnoho potencionálních řešitelů odradit. Prohlížeč má dnes k dispozici každý.

Výsledná aplikace byla otestována nejdříve interně v rámci skupiny BUT Speech@FIT, zapracoval jsem tedy feedback mířený především na uživatelské rozhraní. A poté byla otestována přímo návštěvníky portálu SuperLectures.com. Našli se dobrovolníci, kteří jsou ochotni vypomoci i bez finanční motivace, ale jejich počet není takový, jaká byla původní představa. Spousta návštěvníků na práci pouze nahlédne a tím to pro ně končí. Crowdsourcing je silný nástroj pro zvládnutí výše požadovaného, ale je nutné veřejnost správně motivovat. Ze sledování činnosti těch, kteří opravdu pracují, vyšlo najevo, že jim uživatelské rozhraní vyhovuje a chápou, co se od nich žádá.

Aplikace byla vytvářena iterativně a požadavky na funkčnost aplikace se průběžně měnily. V rámci této práce je tedy Anotační portál úzce spjat přímo s portálem SuperLectures.com, ale systém správy prací je připraven k rozšíření o další způsoby přiřazování práce a typy vykonávané práce.

Do budoucna hodlám ve spolupráci s výzkumnou skupinou BUT Speech@FIT na Anotačním portálu pokračovat a naplno využít potenciál tohoto nápadu.

Literatura

[1] Andi GUTMANS, D. R., Stig Saether BAKKEN: *Mistrovství v PHP5*. Computer Press, a.s., druhé vydání, 2008, ISBN 80-251-0799-X.

[2] BERNARD, B.: Úvod do architektury MVC. [online], 5 2009, [rev, 2009-05-07], [cit. 2012-05-05].
URL http://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/

[3] CHUDADA, M.: *Návrh moderních webů s integrací Web 2.0 služeb*. bakalářská práce, Masarykova univerzita, Fakulta informatiky, Brno, 2009, vedoucí práce doc. RNDr. Tomáš Pitner, Ph.D.

[4] Estellés-Arolas, E.; González-Ladrón-De-Guevara, F.: Towards an integrated crowdsourcing definition. *J. Inf. Sci.*, ročník 38, č. 2, Duben 2012: s. 189–200, ISSN 0165-5515, doi:10.1177/0165551512437638.
URL http://dx.doi.org/10.1177/0165551512437638

[5] Evans: *Domain-Driven Design: Tacking Complexity In the Heart of Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003, ISBN 0321125215.

[6] Fowler, M.: *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002, ISBN 0321127420.

[7] HOWE, J.: The Rise of Crowdsourcing. [online], 06 2006, [rev, 2006-06-XX], [cit. 2012-04-29].
URL http://www.wired.com/wired/archive/14.06/crowds.html

[8] Howe, J.: *Crowdsourcing: Why the Power of the Crowd Is Driving the Future of Business*. New York, NY, USA: Crown Publishing Group, první vydání, 2008, ISBN 0307396207, 9780307396204.

[9] HROMÁDKOVÁ, P.: *Multiplatformní přístup k databázovým aplikacím: Multiplatform Access for Databases Applications*. diplomová práce, Vysoké učení technické, Fakulta informačních technologií, Brno, 2008, vedoucí práce Jiří Kopecký.

[10] KAŠPAR, J.: *Softwarové architektury a návrhové vzory ve webových aplikacích: Software Architecture and Design Patterns in Web Applications*. bakalářská práce, Vysoké učení technické, Fakulta informačních technologií, Brno, 2008, vedoucí práce Dušan Vrážel.

[11] LAVIN, P.: *PHP - objektově orientované, koncepty, techniky a kód*. Praha: Grada Publishing, a.s., první vydání, 2009, ISBN 978-80-247-2137-8, 224 s.

[12] LEHKÝ, P.: *Technologie tvorby webových aplikací*. diplomová práce, Masarykova univerzita, Fakulta informatiky, Brno, 2007, vedoucí práce Mgr. Tomáš Staudek, Ph.D.

[13] NATIONS, D.: What is Web Application? [online], [cit. 2012-04-28].
URL
http://webtrends.about.com/od/webapplications/a/web_application.htm

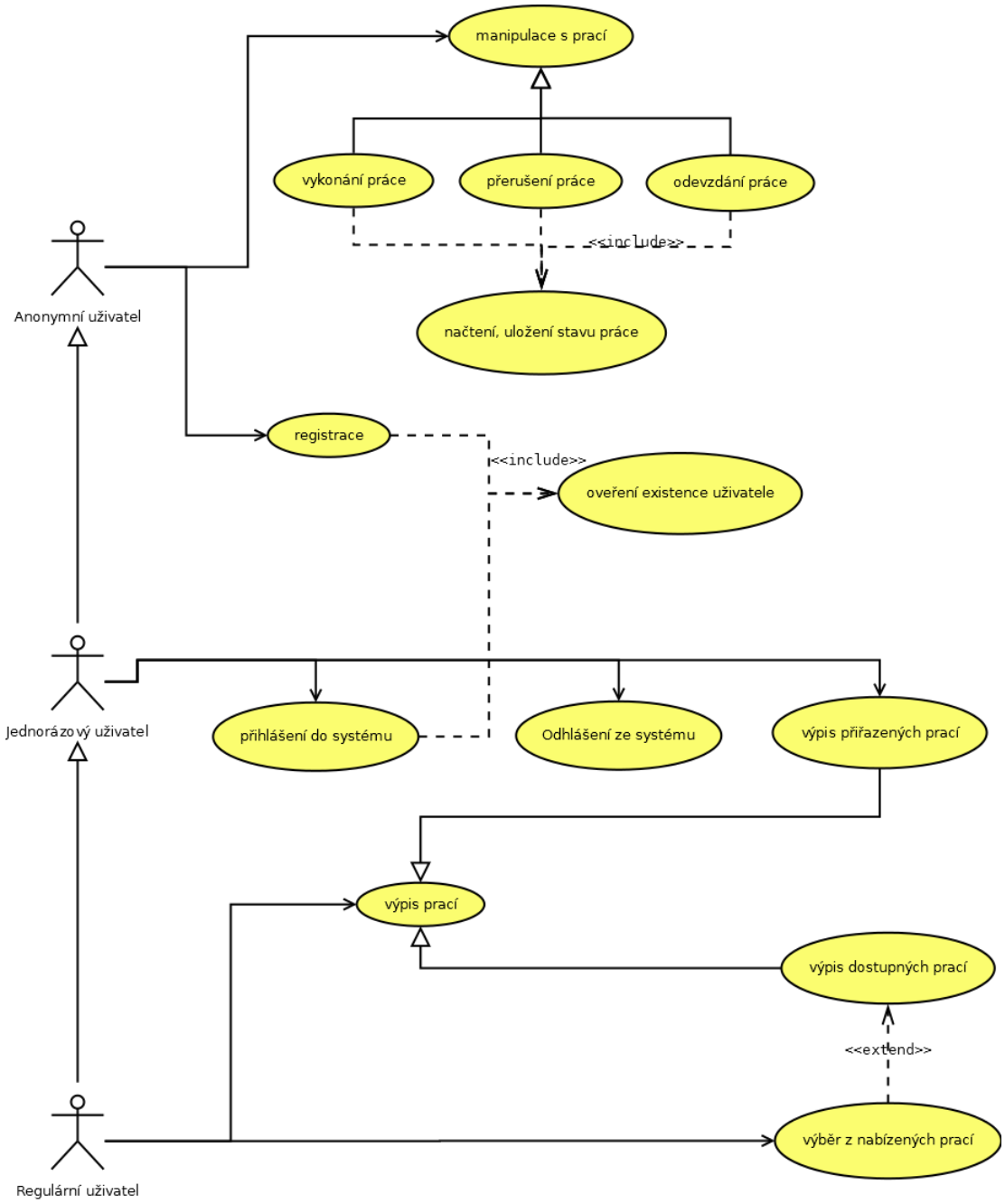
[14] SOVOVOVÁ, E.: Crowdsourcing a jeho využití v praxi. [online], 02 2010, [rev, 2010-02-02], [cit. 2012-04-29].
URL http://www.inflow.cz/crowdsourcing-jeho-vyuziti-v-praxi

[15] SPICK, N.: Towards a WebOS. [online], 09 2007, [rev, 2007-09-XX], [cit. 2012-04-30].
URL http://novaspivack.typepad.com/nova_spivacks_weblog/2007/02/steps_towards_a.html

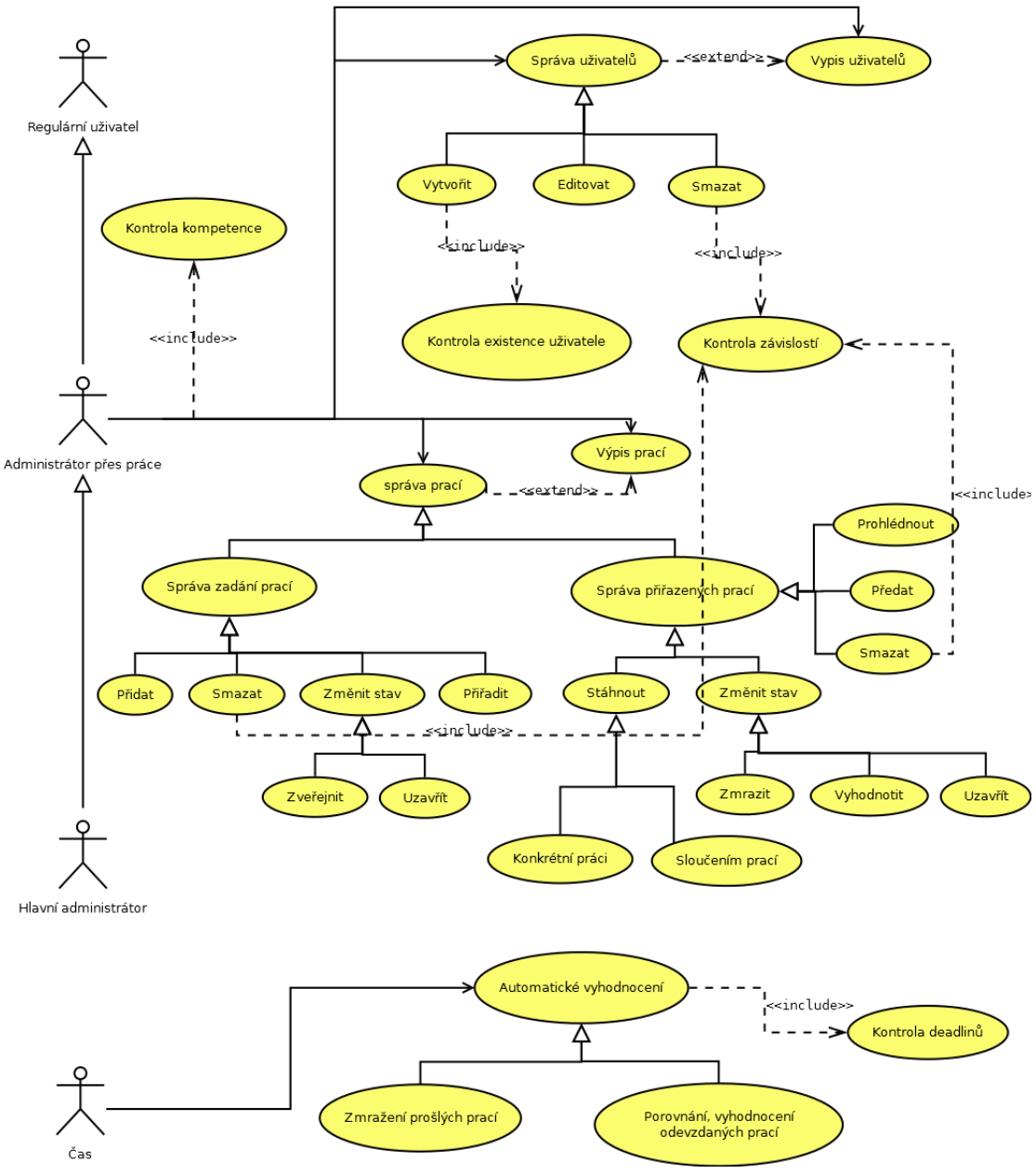
[16] TICHÝ, J.: Pět vrstev modelu. [online], 4 2010, [rev, 2010-04-27], [cit. 2012-05-05].
URL http://www.phpguru.cz/clanky/pet-vrstev-modelu

Příloha A

Diagramy případů užití



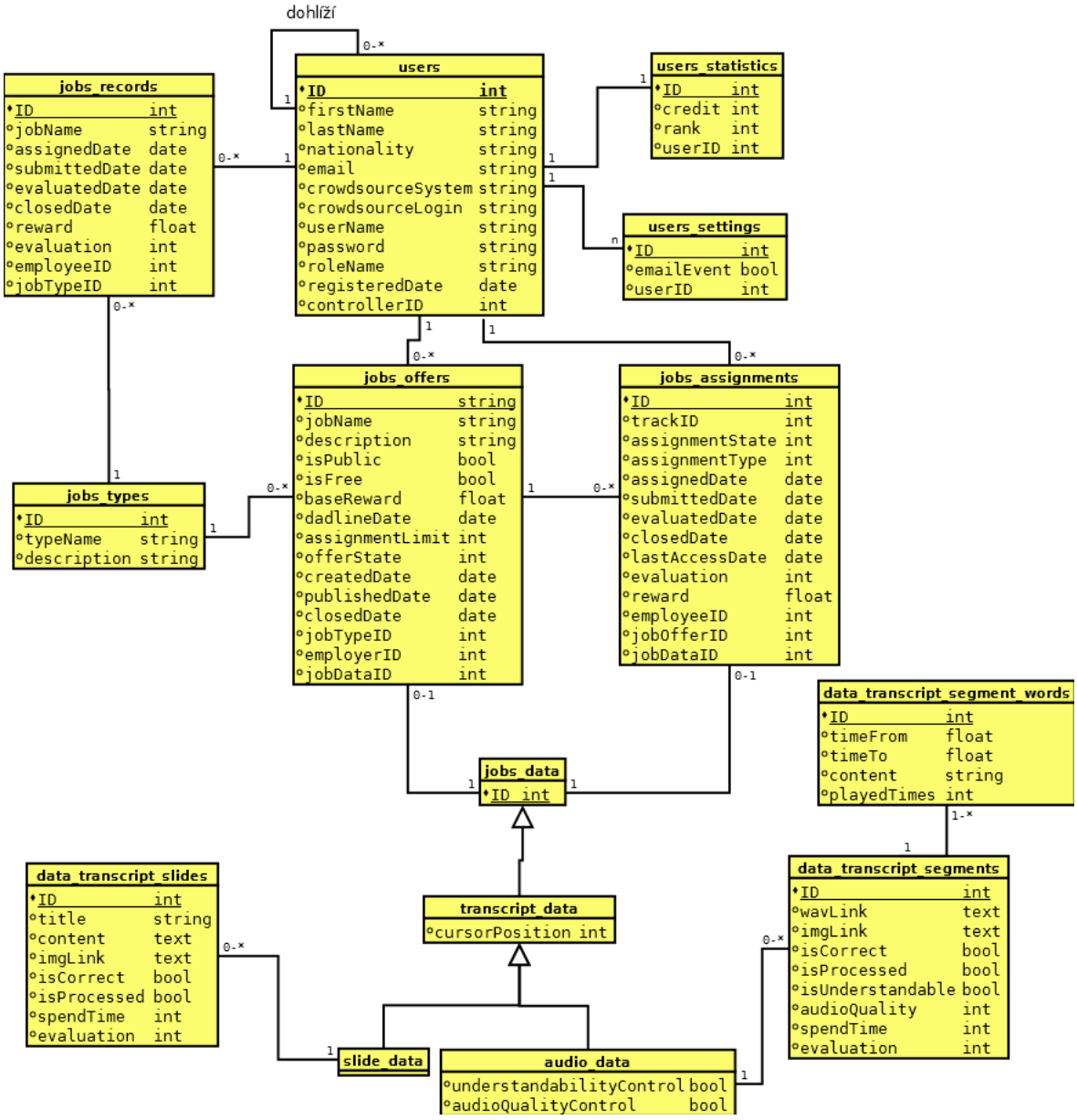
Obrázek A.1: Diagram případů užití zahrnující běžné uživatelské role.



Obrázek A.2: Diagram případů užití zahrnující administrátorské role.

Příloha B

Entity-Relationship diagram



Obrázek B.1: Konceptuální schéma databáze zobrazené pomocí Entity-Relationship diagramu.

Příloha C

Formát XML pro import a export

```
<?xml version="1.0" encoding="utf-8"?>
<data>
	<common name="" assignmentLimit="" baseReward="" deadlineDate="" public="" free="">
		<description>
			common description
		</description>
	</common>
	<job id="" name="" type="">
		<description>
			overloaded description
		</description>
		<items>
			<slides>
				<slide src="" title="">
				<content>
				</content>
				</slide>
			</slides>
		</items>
	</job>
</data>
```

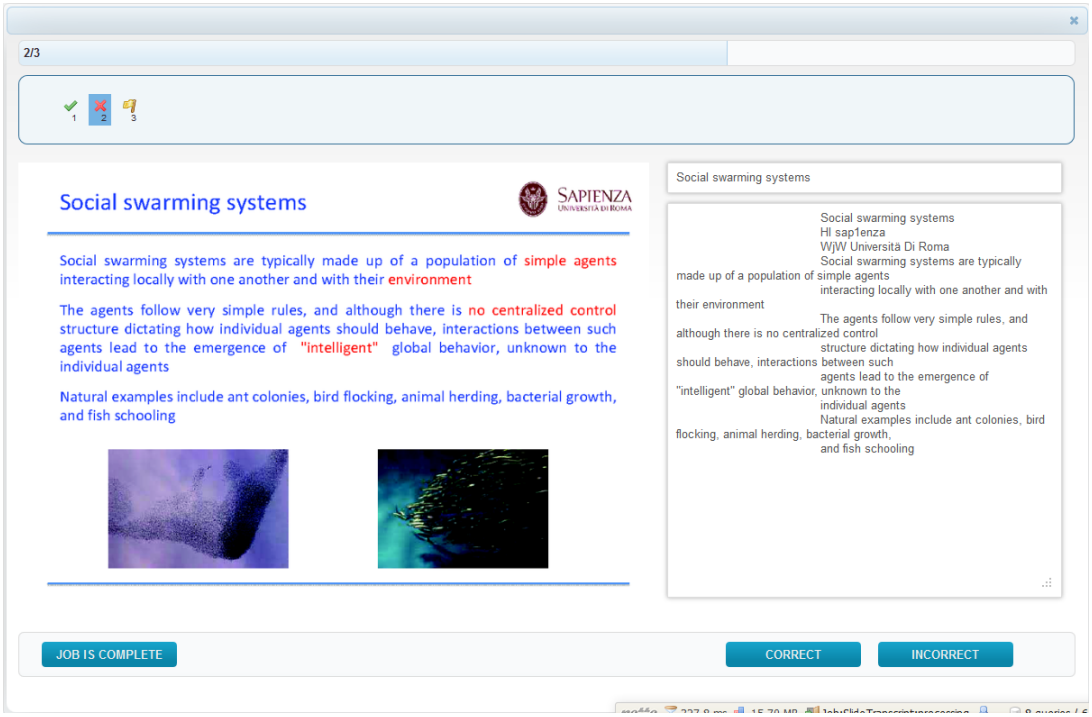
Ukázka kódu C.1: Formát XML pro import/export dat práce typu kontrola, oprava přepisů slajdů.

```
<?xml version="1.0" encoding="utf-8"?>
<data>
	<common type="" assignmentLimit="" baseReward="" deadlineDate="" public="" free="">
		<description>
			common description
		</description>
	</common>
	<job id="" name="">
		<description>
			individual description
		</description>
		<items>
			<segments>
				<segment src="" img="">
					<word from="" to=""></word>
				</segment>
			</segments>
		</items>
	</job>
</data>
```

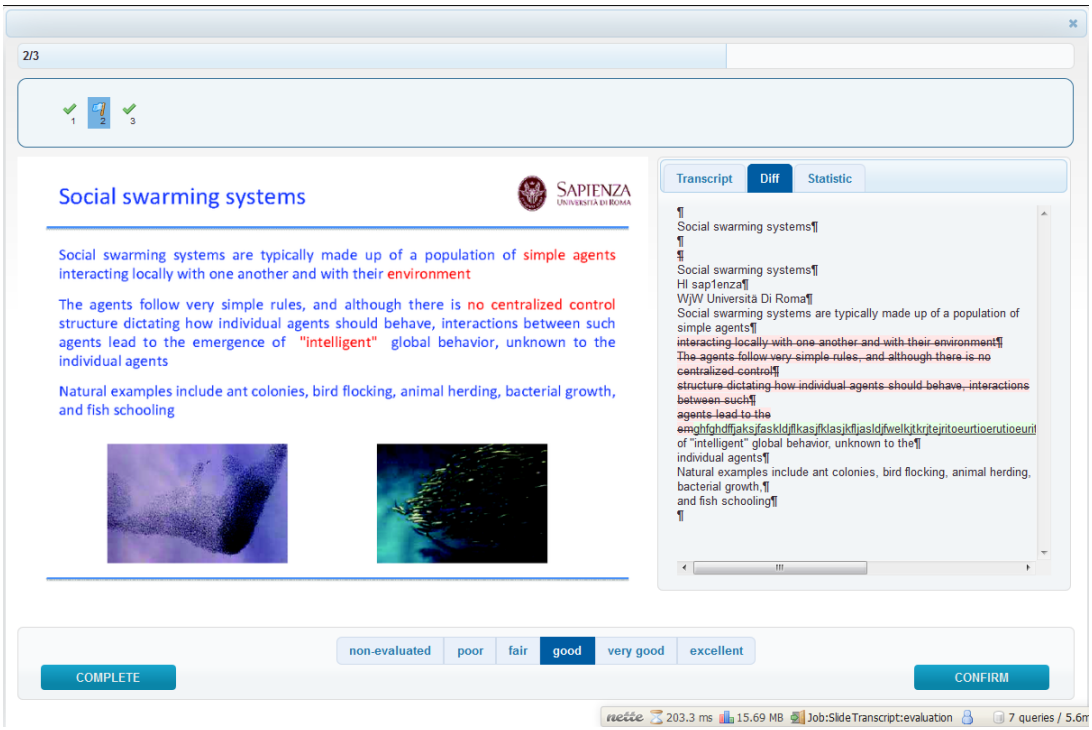
Ukázka kódu C.2: Formát XML pro import/export dat práce typu kontrola, oprava přepisů audia.

Příloha D

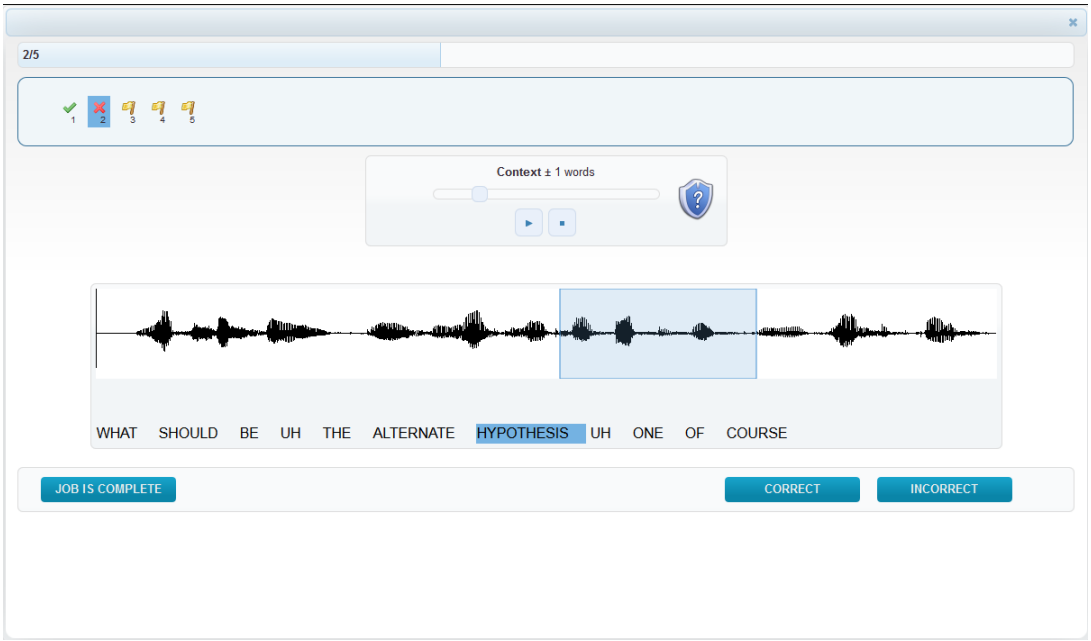
Ukázky uživatelského rozhraní



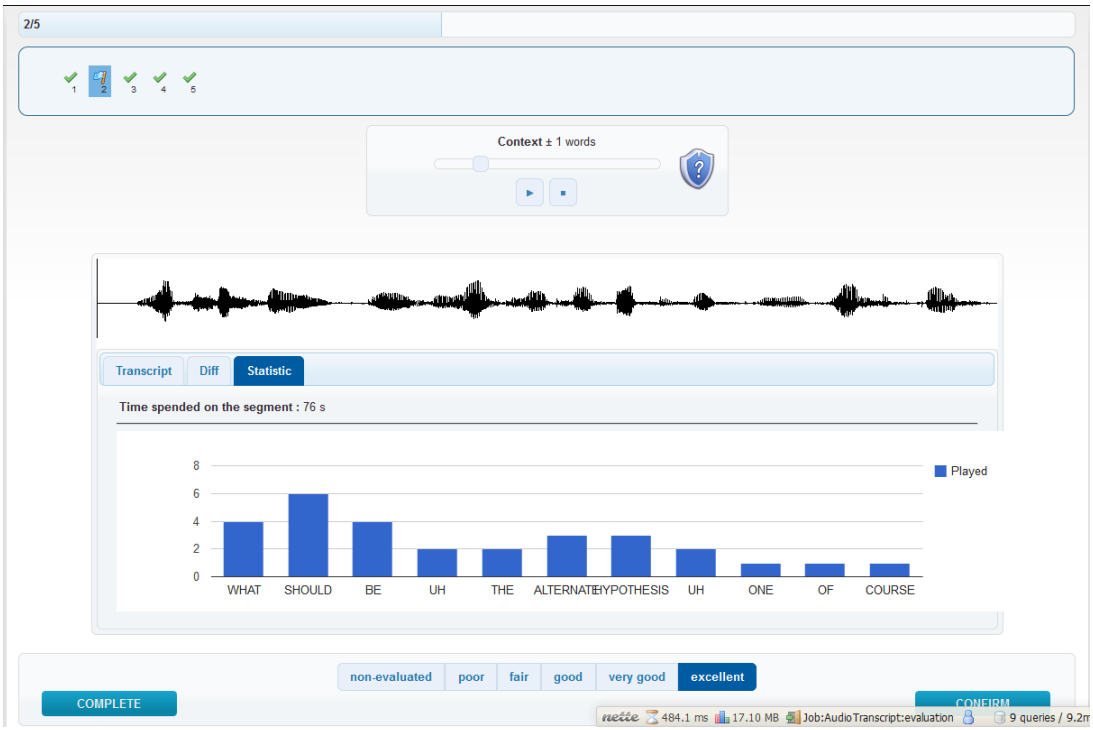
Obrázek D.1: Uživatel má k dispozici *miniaturní seznam všech položek ke zpracování*, ten mu vizualizuje aktuální pozici a stav položky, může se kliknutím na konkrétní položku přesouvat. Pro označení správnost slouží tlačítka *correct a incorrect*, po dokončení práci odevzdává přes *job is complete*.



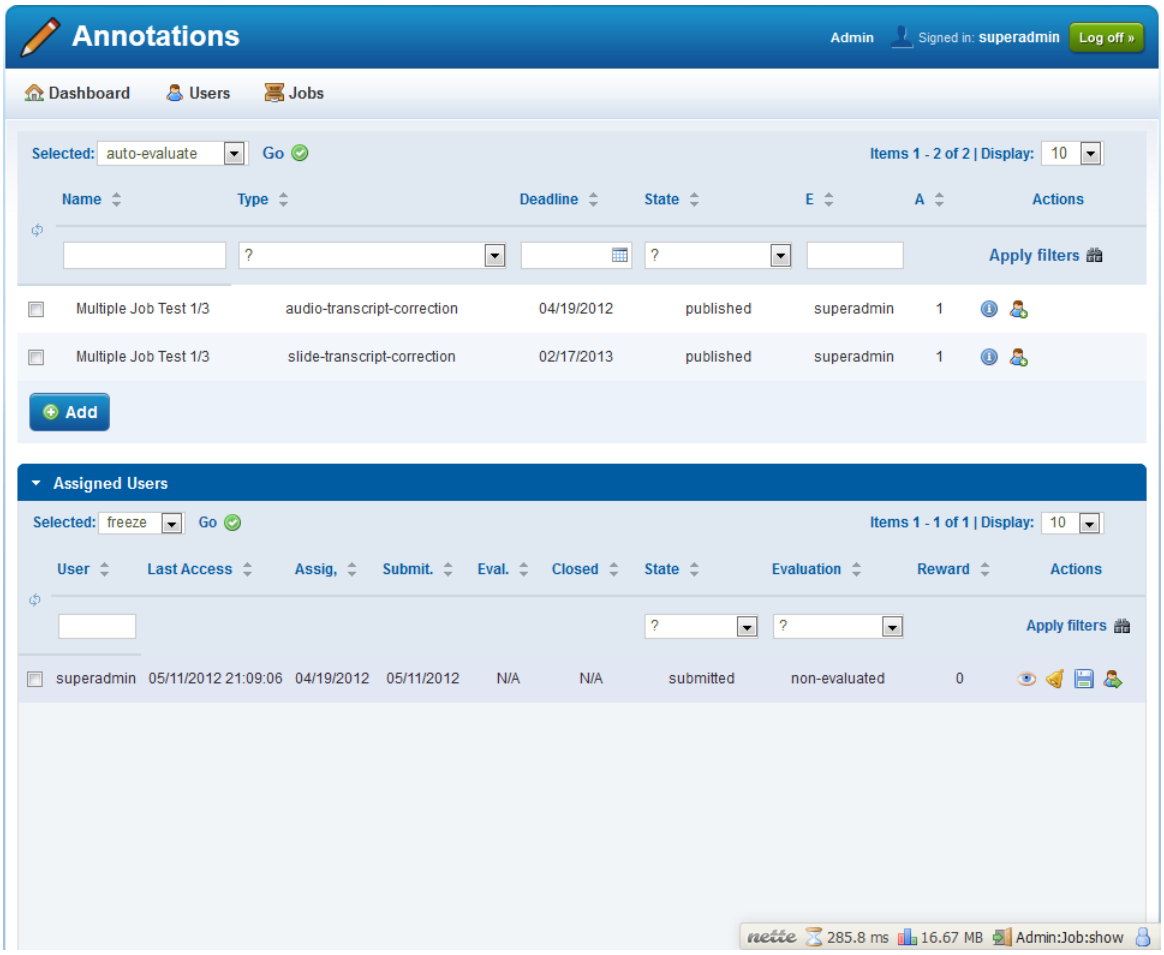
Obrázek D.2: Kromě vlastního přepisu má administrátor po ruce grafický rozdíl změn přepisu vůči zadání a statistiku. Na škále vybírá odpovídající hodnocení a potvrzením se přesouvá na další položku. Ohodnocená položka je vizualizována v seznamu. Po dokončení dává *complete* a tím manuální hodnocení uzavírá.



Obrázek D.3: Způsob pohybu mezi položkami a odevzdání práce je stejný jako u přepisů slajdů. Navíc si může uživatel přehrávat zvukovou stopu pomocí klasického ovládání nebo klávesových zkratek. Pomocí nich nebo kliknutím do obrázku na požadované slovo si jej může přehrát se zvoleným kontextem. Využití klávesových zkratek dokáže uživatele naprosto oprostit od použití myši.



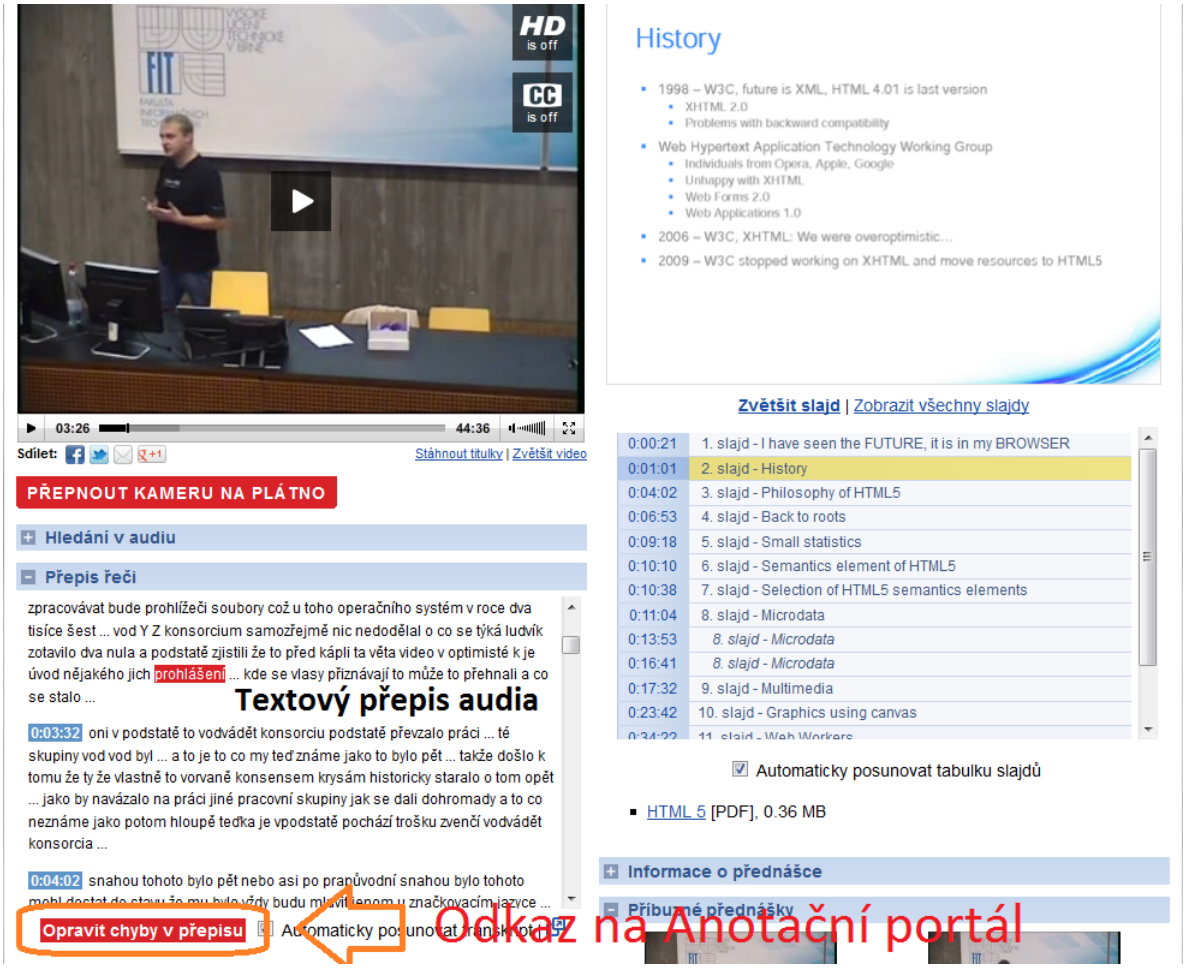
Obrázek D.4: Podobně jako u vyhodnocení oprav přepisů slajdů zde má administrátor vlastní přepis, jeho rozdíl vůči zadání a statistiky. Z těch může vyčíst, jaký čas uživatel na segmentu strávil a počet přehrátí každého slova. Pro ověření správnosti přepisu si může přehrávat zvukovou stopu.



Obrázek D.5: V první polovině má administrátor přehled o všech zadaných pracích v systému, pokud na to má pravomoce. Ve druhé polovině může nahlížet na přiřazení konkrétní práce.

Příloha E

Integrace v portálu SuperLectures.com



Obrázek E.1: Narazí-li návštěvník na chybu v přepisu, má možnost daný segment přednášky opravit.